PROYECTO FINAL DE CARRERA

ESTUDIO DEL ALGORITMO DE DECODIFICACIÓN MLLA EN ENTORNOS FINGERPRINTING

Estudios: Enginyeria de Telecomunicació

Autor: Ana Mª Gómez Muro

Director: Joan Tomàs Buliart

Fecha:Octubre 2010

"La verdadera ciencia enseña, por encima de todo, a dudar y a ser ignorante."

Miguel de Unamuno

RESUMEN

EL PRINCIPAL OBJETIVO DE este proyecto es encontrar una posible solución a los ataques de confabulación puesto que estos constituyen el principal problema del fingerprinting digital. Con este fin se ha estudiado el rendimiento del algoritmo MLLA en determinados esquemas de fingerprinting.

Primero se ha implementado el algoritmo Max-Log-MAP y se ha comprobado su correcta implementación con diversas simulaciones. Se ha modificado la relación señal/ruido introducida por el canal AWGN hasta obtener el umbral a partir del cual la decodificación era correcta. Se ha calculado su tiempo de ejecución obteniendo una clara mejora respecto al tiempo del algoritmo MAP. Por último, se ha probado su funcionamento en esquemas de fingerprinting donde dos usuarios realizan un ataque por confabulación, y el decodificador siempre retornaba una lista cuyas dos primeras palabras correspondían a los dos usuarios que formaban parte de la coalición.

Después se ha implementado el algoritmo MLLA. Se ha verificado su correcto funcionamiento usando un codificador/decodificador convolucional y un canal AWGN, comprobando que la primera palabra de la lista retornada por el algoritmo MLLA siempre es la palabra enviada. Se ha buscado el rendimiento del algoritmo MLLA en entornos de fingerprinting donde dos usuarios realizan un ataque por confabulación, en diferentes escenarios. Según el tamaño de la lista de posibles secuencias de información enviada que retorna el decodificador. Según la longitud de la palabrada enviada a traves del turbo codificador. Según la longitud del trellis utilizado en turbo codificación. Y finalmente, introduciendo el método de verificación de redundancia cíclica.

A grandes rasgos, las probabilidades de encontrar a alguno de los usuarios que participan en una coalición no han tenido los altos valores que se deseaban. Sin embargo, tras el análisis de los resultados obtenidos, ésta puede ser una primera aproximación que nos acerque a la solución del problema inicialmente planteado a la que se le deben de introducir mejoras en cuanto al rendimiento.

AGRADECIMIENTOS

DESPUÉS DE DEDICAR TANTO TIEMPO a la realización de un trabajo, son muchas las personas que se han subido conmigo a esta particular noria de emociones. Para todas ellas, únicamente tengo palabras de agradecimiento.

A Joan Tomàs Buliart, por todas las horas dedicadas a soportar mis preguntas y dudas constantes, por las soluciones aportadas y por los ánimos en momentos de frustración. A Marcel Fernández Muñoz, por dedicar su tiempo a aportar luz en momentos de oscuridad. A Miquel Soriano Ibáñez, por ponerme en contacto con Joan Tomàs haciendo posible mi participación en la realización de este proyecto.

A todas la personas que compartieron conmigo la experiencia de la UPC, a los que pasaron y a los que se han quedado. A Lorena, Guti, Mireia y Luis, por tantos momentos y esas experiencias inolvidables que fueron México y Venezuela. A Rubén y Javi por escuchar tantas veces mis quejas, por ser tan comprensivos y no enviarme bien lejos. A todos los que no compartieron estudios pero se cruzaron en mi camino y compartieron dudas y momentos para no olvidar.

A Quique, Caetano, Pedro, Andrés, Jorge, Micah, Nacho, Enric, Lila, Patrick, Christina, Astrud, Xoel, Iván, Javier, Carmen, Joaquín, Ramón, Chavela, Marcos, Kevin, Diego, Albert, Eva, Paulinho, Antonio, Kiko y muchísimos más. Porque sin su compañía y sabiduría jamás habría hecho este trabajo.

A Cristina, Esther y Miriam, las tres patas que conforman conmigo la base de una mesa *vintage*. Mesa que ha sufrido y disfrutado de manera desmesurada y que, día a día, procuro cuidar como pieza de museo.

A la constante preocupación de mis tíos y primos, por estar siempre aquí a pesar de vivir allí. A la *Mista* porque, aunque no entienda qué hace su nieta, es la abuela más moderna del mundo. A los que se fueron y estarían tan contentos.

A Julián, por sus contínuos empujones, por ser tan de letras y darme siempre la clave, *sim-pli-fi-car*. A Sílvia, por no ser tan de ciencias y comprender mis locuras.

A mis incondicionales *Viendre* y *Jubileta*, por esperar a que el autobús pasara por este callejón sin salida. Por soportar mi inquieta espera, haciéndola mucho más llevadera con nuestro particular idioma y las escapadas en bici. Por vivir de manera descomedida mis sufrimientos y alegrías. Éste es vuestro proyecto.

Muchas gracias a todos

CONTENIDO

1

INTRODUCCIÓN

EN ESTE CAPÍTULO SE EXPONE LA PROBLEMÁTICA QUE DA LUGAR AL ORIGEN DE ESTE PROYECTO Y SE INTRODUCEN LOS CONCEPTOS BÁSICOS QUE EN CAPÍTULOS POSTERIORES SERÁN DESARROLLADOS. FINALMENTE SE DEFINE LA ESTRUCTURA DE ESTE DOCUMENTO DETALLANDO LOS CONTENIDOS PROPIOS DE CADA CAPÍTULO.

1.1 MOTIVACIÓN

EN LOS ÚLTIMOS AÑOS la utilización de soportes digitales (DVD, CD, ficheros, ...) ha aumentado vertiginosamente. De la misma manera han proliferado diferentes formas de que un usuario se haga con una copia ilegal de un determinado material. Por un lado, los dispositivos usados para crear duplicados de contenidos digitales se han extendido hasta el punto de que cualquier persona puede duplicar en su casa un DVD o un CD de manera sencilla. Además, el uso de Internet, sobretodo mediante programas P2P, ha permitido que millones de usuarios usen la red para compartir material audiovisual.

Hay dos posibilidades de evitar que esto suceda: la protección *a priori* y la protección *a posteriori*. La primera consiste en intentar impedir que el cliente pueda realizar una copia del material, mientras que la segunda consiste en detectar

dichas copias. Durante varios años se pensó que la mejor manera de protegerlo era mediante técnicas de protección *a priori*. Con el tiempo se ha demostrado que este tipo de protecciones no son todo lo robustas que se creía en un principio, ya que se puede dar con el algoritmo de protección y, por tanto, anularlo. Así pues, en los últimos años se está trabajando en la protección *a posteriori* como una herramienta eficiente para combatir la piratería. Ésta consiste en insertar un conjunto de bits (mediante algoritmos de watermarking) en los contenidos del producto de soporte electrónico que se quiere proteger sin que esto se note en el resultado final. Si dichas marcas contienen información del comprador, esto nos permite identificarlo y, por tanto, detectar el responsable de la copia ilegal. Cuando nos encontramos en este caso hablamos de huella digital (fingerprinting).

El problema aparece cuando dos usuarios (lo que se llama coalición) generan una nueva copia a partir de las suyas. Esta nueva copia tiene un fingerprint diferente al de ambos usuarios. Esto provoca que ninguno de ellos resulte identificado o que sea difícil de rastrear. Es decir, que ninguno de los usuarios que participan en la coalición resulte incriminado y puedan distribuir la nueva copia de forma ilícita. A este tipo de ataques se les llama ataques de confabulación y son el principal problema del fingerprinting digital. En la actualidad, la única forma de detectar usuarios que forman parte de una confabulación es realizando una decodificación por fuerza bruta. Este tipo de decodificaciones, dado que utilizan el método de prueba y error, son muy costosas en tiempo computacional. Por tanto, surge la necesidad de intentar buscar códigos fingerprinting que puedan resistir coaliciones entre usuarios.

En este proyecto se pretende encontrar una posible solución a este tipo de ataques. Para ello se ha estudiado la posibilidad de usar el algoritmo de decodificación MLLA para ser usado en este contexto. Una vez implementado dicho algoritmo, se ha hallado su rendimiento en entornos fingerprinting donde dos usuarios realizan ataques de confabulación bajo distintas condiciones.

1.2 CONOCIMIENTOS PREVIOS

PARA LA REALIZACIÓN de este proyecto ha hecho falta un importante trabajo de documentación previo sobre diversos temas para poder proceder a la posterior implementación del mismo. Los temas en los que se ha tenido que ahondar han sido:

- **Turbo Códigos:** su estudio a fondo es imprescindible puesto que son el tipo de códigos usados como base del sistema a implementar.

- **Algoritmo de decodificación Max-Log-MAP:** es un algoritmo clave, usado para decodificar turbo códigos, puesto que ha servido como enlace para pasar de un turbo decodificador implementado con el algoritmo MAP al implementado con el algoritmo de decodificación MLLA.
- **Algoritmo de decodificación MLLA (Max-Log List Algorithm):** el estudio y comprensión de este algoritmo ha constituido la parte del proyecto que mayor número de horas ha conllevado. Se disponía de una única fuente de información sobre su funcionamiento lo que ha supuesto que se hayan tenido que desarrollar múltiples interpretaciones hasta llegar a la correcta.
- **Programación en MATLAB:** para la simulación de los modelos implementados en este proyecto se necesita interactuar con MATLAB. Para trabajar con MATLAB ha sido necesaria una familiarización previa con este programa, su lenguaje de programación y sus librerías.

1.3 OBJETIVOS

LOS OBJETIVOS QUE SE PRETENDEN alcanzar mediante la realización de este proyecto son los expuestos a continuación:

- Realizar la implementación del algoritmo de decodificación Max-Log-MAP, que es la base del algoritmo MLLA, usando un canal AWGN (Additive White Gaussian Noise). Buscar el mínimo umbral de la relación señal/ruido (SNR) a partir del cual dicha implementación decodifica sin ningún error.
- Hallar el rendimiento del algoritmo Max-Log-MAP en entornos de fingerprinting donde haya dos usuarios que se alíen para realizar un ataque de confabulación.
- Realizar la implementación del algoritmo de decodificación MLLA. Comprobar el correcto funcionamiento de éste utilizando un codificador/decodificador convolucional y un canal AWGN.
- Estudiar el rendimiento del algoritmo de decodificación MLLA, en entornos de fingerprinting donde dos usuarios realizan un ataque por confabulación, en función del tamaño de la lista de posibles secuencias de información enviada que es retornada por el mismo. Para ello se usará un turbo codificador/decodificador y un canal AWGN.

- Estudiar el rendimiento del algoritmo de decodificación MLLA, en entornos de fingerprinting donde dos usuarios realizan un ataque por confabulación, en función del tipo de trellis utilizado para la codificación y del tamaño de las palabras codificadas. Para este tipo de simulaciones se utilizará un turbo codificador/decodificador y un canal AWGN. Buscar la probabilidad de culpar a usuarios inocentes, que no han participado en la coalición, introduciendo el método de verificación de redundancia cíclica.

1.4 ESTRUCTURA DE LA MEMORIA

EN EL CAPÍTULO 2 se explica el marco en el que se sitúa este proyecto, es decir, los motivos por los que surge la necesidad de su implementación. En el capítulo 3 se definen los turbo códigos que son el tipo de códigos utilizados para la realización de este proyecto. En el capítulo 4 se hace un estudio detallado del funcionamiento del algoritmo de decodificación MLLA que incluye un ejemplo de este tipo de decodificación. En el capítulo 5 se explica el código realizado para obtener la implementación y se muestran aquellas partes del mismo que más dificultad han representado. En el capítulo 6 se enseñan las simulaciones hechas para verificar el correcto simultado de la implementación y los resultados obtenidos aplicando el algoritmo MLLA en determinados escenarios de fingerprinting. Las conclusiones y líneas de trabajo futuras se discuten en el capítulo 7.

—2—
ESTADO DEL ARTE

EN ESTE CAPÍTULO SE PRESENTAN LOS 3 CONCEPTOS BÁSICOS QUE ENMARCAN EL CONTEXTO DE ESTE PROYECTO: EL *WATERMARKING* (CÓMO INCRUSTAR MARCAS DENTRO DE UN DOCUMENTO DIGITAL), EL *FINGERPRINTING* (CÓMO GENERAR LAS MARCAS PARA QUE TENGAN UNA DETERMINADA SERIE DE PROPIEDADES) Y LOS *CÓDIGOS CORRECTORES DE ERRORES* (CÓMO PROTEGER LA INTEGRIDAD DE LA INFORMACIÓN).

2.1 DIGITAL WATERMARKING

LA DEFINICIÓN DE *watermark* es simple. El watermark o marca de agua es un mensaje que se introduce en un documento y que aporta información sobre el autor o la obra. Esta definición tan ambigua agrupa una cantidad importante de campos y matices. Desde el soporte (audio, vídeo, texto, ...) hasta la tipología de la marca hay un amplio abanico de posibilidades.

No debe confudirse la criptografía con el watermarking. Cuando se cifra un contenido se está forzando a que el usuario o cliente disponga de una determinada clave específica para ese contenido (el término clave no debe entenderse únicamente en el entorno criptográfico, se entiende clave como un software específico, un código, ...). En el watermarking no, es decir, no se intenta privar a ningún usuario de la utilización de un contenido, lo que se pretende es impedir que un usuario deshonesto elimine una información determinada (de propiedad o de autoría, por ejemplo) del documento.

En otras palabras, tomando como ejemplo un correo electrónico, cifrarlo evitaría que cualquiera pudiera leer el correo, en cambio, el watermarking permitiría evitar que un usuario deshonesto borrara la firma del final del correo y realizara un reenvío del mismo atribuyéndose la autoría. Este ejemplo no pretende tener sentido práctico, pero sí clarificar la diferencia entre protección criptográfica y watermarking.

El objetivo técnico principal de los esquemas de digital watermarking es construir un buen watermark. Para conseguirlo se tienen que fijar los requerimientos básicos que hacen que un watermark se considere bueno. Estos requerimientos son:

1. **Fidelidad:** Un watermark tiene que ser perceptiblemente invisible, es decir, tendría que causar una degradación nula del contenido original.

2. **Robustez:** Un watermark tiene que ser difícil de eliminar. Particularmente, tiene que ser resistente a las distorsiones provocadas por los típicos procesados de señales (como conversiones de digital-analógico y analógico-digital, recuantificación, recompresión, ...) y a las distorsiones geométricas habituales (como rotaciones, traslaciones, escalados, ...).

3. **Capacidad:** Un sistema de watermarking tiene que ser capaz de incrustar cantidades relativamente grandes de información ya que cuanta más capacidad se tenga, más flexibilidad y usos se le podrá dar al esquema.

Algunos de estos requerimientos entran en conflicto entre ellos. Por ejemplo, un sistema de watermarking diseñado para presentar una robustez muy elevada provocará de forma inevitable una distorsión elevada del documento marcado. En cambio, un watermark invisible y robusto difícilmente podrá ofrecer una capacidad elevada. Como resultado, el diseño de un esquema de watermarking es un compromiso entre estos requerimientos y, a la vez, estos requerimientos serán más o menos flexibles en función de la aplicación a la que se destine el watermark. Un ejemplo es el llamado watermarking frágil diseñado para detectar cualquier manipulación producida en un documento. En este caso, la robustez está claro que es una propiedad no deseada ya que lo que se pretende es precisamente que el watermarking sea fácilmente eliminable o, en este caso, alterable.

Además existen otros requerimientos complementarios y que varían mucho en función de la aplicación. Los más relevantes son los siguientes:

1. **Indetectable:** es la imposibilidad de probar la presencia de un mensaje oculto o de una marca. Este concepto está fuertemente ligado al modelo estadístico

del documento original. Se debe remarcar que la capacidad de detectar la presencia de una marca no implica directamente la habilidad de eliminarla, pero en muchos casos la aplicabilidad de un esquema en un determinado entorno puede depender de este requerimiento. Muchas veces se confunde esta propiedad con la de fidelidad. La fidelidad sólo responde a requerimientos perceptuales, en cambio, esta propiedad contempla el análisis estadístico de un sistema.

2. **Complejidad:** el proceso de generación de una marca no puede ser trivial ya que esto podría comportar que un atacante pudiera llegar al extremo de falsificar una marca y que el sistema no fuera capaz de diferenciar la falsificación de la marca original.

3. **Clave de acceso:** la información incrustada no puede ser extraída ni siquiera con ataques diseñados conociendo el algoritmo de incrustación y de extracción (excepto una clave secreta) y el conocimiento de, como mínimo, un documento marcado y su marca. Esta propiedad es la misma que se pide habitualmente a los sistemas criptográficos, en los que el algoritmo depende de una información secreta de la que sólo disponen los usuarios autorizados.

4. **Baja probabilidad de error:** un factor importante es minimizar la probabilidad de que una marca se pueda detectar de forma incorrecta. Se tiene que entender que este hecho es diferente a no detectar la marca. Es decir, no detectar la marca significa que el algoritmo de recuperación es incapaz de encontrar una marca válida. Como contraposición, detectar una marca de manera incorrecta quiere decir que se recupera una marca válida pero diferente a la marca que se había incrustado originariamente. Este factor puede provocar la inculpación de un usuario honesto. Este tema es muy delicado y es el motivo por el cual muchos algoritmos han sido descartados.

5. **Coste computacional en la inserción y en la extracción:** A pesar de que normalmente los ficheros de entrada tienen medidas considerables, se tiene que intentar que los algoritmos tengan un coste computacional razonable. Este factor puede ser el que actualmente recibe menos atenciones dado el aumento de la potencia de proceso que hoy en día se está produciendo (teniendo en cuenta que cada año se dobla). A pesar de todo, también es importante tenerlo en cuenta de cara al futuro.

2.2 DIGITAL FINGERPRINTING

EL OBJETIVO BÁSICO DEL FINGERPRINTING es proteger contenido con copyright contra distribuciones ilícitas. En realidad la forma de uso es parecida a la empleada en el campo de la investigación forense con las huellas digitales. En este caso, el hecho de no poder identificar a un individuo mediante sus huellas no evita que este individuo cometa su acto delictivo. A pesar de todo, lo disuade de que lo haga ya que sabe que la autoridad dispone de técnicas para obtener pruebas irrefutables de que ha sido él. De la misma manera, el fingerprinting digital *per se* no evita que se produzcan distribuciones ilegales, pero busca disuadir que se hagan proporcionando herramientas para identificar sin dudas razonables a quien lo haga.

El funcionamiento es sencillo, un distribuidor adquiere un determinado contenido y los derechos de distribución del mismo. Este distribuidor tiene una serie de clientes interesados en este contenido y llegan a un acuerdo comercial para adquirir copias de este contenido. Cada copia se identifica con una marca (o *fingerprint*) formada por un conjunto de dígitos redundantes que son incrustados dentro de esta copia mediante alguna técnica de watermarking. La localización de estos dígitos sólo es conocida por el distribuidor y, por tanto, se mantiene oculta a los clientes (sería equivalente a la clave privada si se hiciera una analogía con el paradigma de la criptografía de clave pública). Las posiciones de estos dígitos son las mismas para todos los clientes variando sólo el valor. De esta manera, si un cliente decide redistribuir su copia, éste podrá ser identificado fácilmente.

El problema aparece cuando no es un único usuario sino que son varios (lo que se llama coalición [1]) los que intentan generar una nueva copia que no identifique a ninguno de ellos o que sea difícil de rastrear. Por tanto, cuando se investiga sobre fingerprinting digital se pretende dar solución a los ataques de confabulación o, en otras palabras, se buscan códigos fingerprinting capaces de resistir coaliciones de un determinado número de usuarios.

Con el fin de realizar implementaciones efectivas de técnicas de fingerprinting digital se tienen que contemplar dos puntos muy importantes: incrustar el código fingerprinting de forma fiable y escoger el código fingerprinting apropiado.

[1] Coalición: grupo de usuarios cuyos miembros intentan atenuar o eliminar los fingerprints de sus copias para generar una nueva copia con un fingerprint diferente a los suyos y que no les pueda incriminar. Las acciones pueden ir desde la media entre copias y distribuirse el resultado, hasta técnicas mucho más sofisticadas.

2.2.1 PROPIEDADES DE LOS CÓDIGOS FINGERPRINTING

Con tal de comparar el rendimiento de los códigos fingerprinting se necesitan algunos criterios de medida, más o menos objetivos, y para definir criterios de medida hay que definir las propiedades a medir. Por tanto, lo primero que se debe hacer es definir las propiedades básicas que se desean para un buen código fingerprinting:

1. **Elevada cardinalidad del *codebook*:** el código tiene que poder tener un gran número de usuarios. A pesar de que en algunas aplicaciones un *codebook* pequeño puede ser adecuado (como es el caso de los jurados de los Oscar que únicamente necesitan algunos miles de copias para los miembros de la Academia), en general, se desea que la medida del *codebook* pueda ser grande. Por ejemplo, el número de copias de una película que son distribuidas a los usuarios finales puede ser del orden de magnitud de millones.

2. **Palabras código cortas:** los sistemas de incrustación imponen una limitación en la capacidad de información que pueden incrustar a un determinado documento. Si las palabras código son cortas, el código fingerprinting será adaptable a un mayor número de aplicaciones que en el caso de que sean largas, ya que no siempre se dispone de un documento con una capacidad muy grande.

3. **Facilidad de rastreo:** cuanto más eficiente sea el algoritmo de rastreo, mejor será el esquema de un determinado código fingerprinting. En el caso más optimista en el que el detector sólo tenga limitada la capacidad computacional, el algoritmo de rastreo debe ser lo suficiente eficiente como para dar la decodificación en un tiempo razonable.

4. **Baja probabilidad de error:** es obvio que una propiedad importante es que el algoritmo de rastreo falle en una probabilidad muy baja y, a poder ser, en caso de fallar que dé signos evidentes de haberse producido un error en lugar de producir falsos positivos, es decir, de inculpar inocentes.

A simple vista se puede observar que algunas propiedades entran en conflicto entre ellas rápidamente. Por ejemplo, la longitud de las palabras código y la medida del *codebook*, ya que normalmente, cuanto más cortas son las palabras código, menor es la cardinalidad. Otro ejemplo es la relación entre la medida de las palabras código y la probabilidad de error, ya que para reducir la probabilidad de error a un determinado nivel, la longitud de las palabras código tendrá que aumentar para añadir redundancia de algún tipo. En todo caso, habitualmente, estas condiciones comportan un aumento del coste computacional de los algoritmos de rastreo. Por

tanto, en muchos casos, cuando se diseña un código fingerprinting se hace necesario tener muy presente la aplicación a la que se destinará.

2.3 CÓDIGOS CORRECTORES DE ERRORES

A LA HORA DE TRANSMITIR UN MENSAJE en código binario pueden aparecer interferencias que eviten que éste sea recibido exactamente tal y como fue enviado: un cero podría ser recibido como un uno, y un uno podría ser recibido como un cero.

Cuando se detecta un error, una posible solución es que el receptor solicite al emisor la repetición del bloque de datos transmitido. Esta técnica se denomina ARQ, Automatic Repeat Request.

Sin embargo, existen algunas aplicaciones en las que no es posible pedir la retransmisión de los datos (resulta poco eficiente o incluso supone un alto gasto económico), por lo que el mensaje debe ser corregido de alguna forma en el destino. Es en estas situaciones cuando se utilizan los denominados Códigos Correctores de Errores (ECC, Error Correcting Code). Esta técnica de corrección de errores es denominada FEC, Forward Error Correction.

2.3.1 MODELO GENERAL DE UN SISTEMA DE COMUNICACIÓN

El modelo general de un sistema de comunicación se muestra en la figura 2.1. En él se pueden distinguir los siguientes elementos:

- Una *fuente de información* que genera una cadena o palabra de longitud k con símbolos o letras en un alfabeto.
- Un *proceso de codificación* que transforma unívocamente el mensaje anterior en otro de longitud $n > k$, sobre el mismo alfabeto u otro diferente, y al que se ha añadido información redundante suficiente como para poder detectar y corregir un número razonable de errores que pudieran producirse durante el proceso de almacenamiento o de transmisión.
- Un *canal* a través del cual se transmite el mensaje anteriormente codificado o en el cual se almacena dicha información, la cual puede sufrir algunos errores debidos al ruido existente en dicho canal.
- Un *proceso de decodificación* que asigna al mensaje distorsionado por el canal otro mensaje que, en caso de haberse producido como máximo un número

determinado de errores, es el mensaje introducido inicialmente en el canal, permitiendo así recuperar la información transmitida.

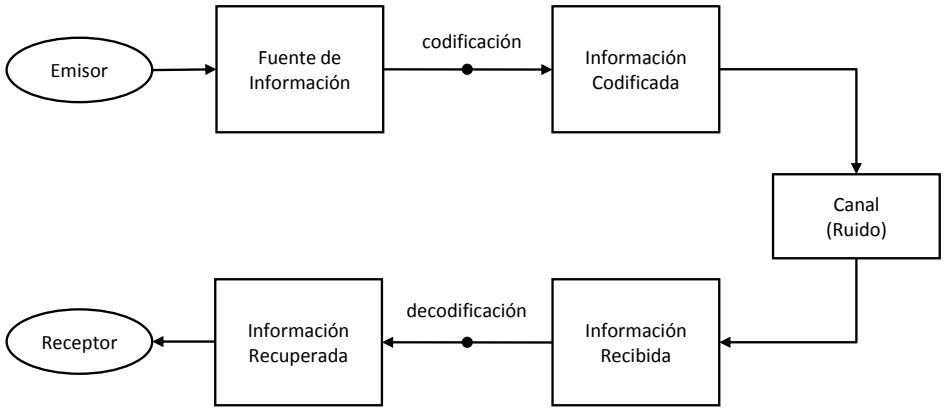

FIGURA 2.1: Modelo general de un sistema de comunicación.

2.3.2 TÉCNICAS DE TRATAMIENTO DE ERRORES

Los errores en la transmisión de información a través de un canal pueden producirse debido a diferentes tipos de ruido:

1. **Aleatorio (ruido blanco o gaussiano):** errores que se producen y presentan aislados.

2. **Ráfagas:** consiste en un gran número de errores consecutivos.

3. **Desvanecimiento.**

Cada tipo de error requiere un tipo de código específico para ser detectado o corregido. De esta manera, podemos clasificar los códigos correctores de error en:

1. **Códigos de bloque:** Estos códigos utilizan series de n símbolos que se designan con el nombre de palabras o codewords. Los n bits que forman una palabra son exclusivamente función de los bits del mensaje actual, y no de los anteriores mensajes. Este tipo de códigos se utilizan en canales con ruido a ráfagas o desvanecimiento.

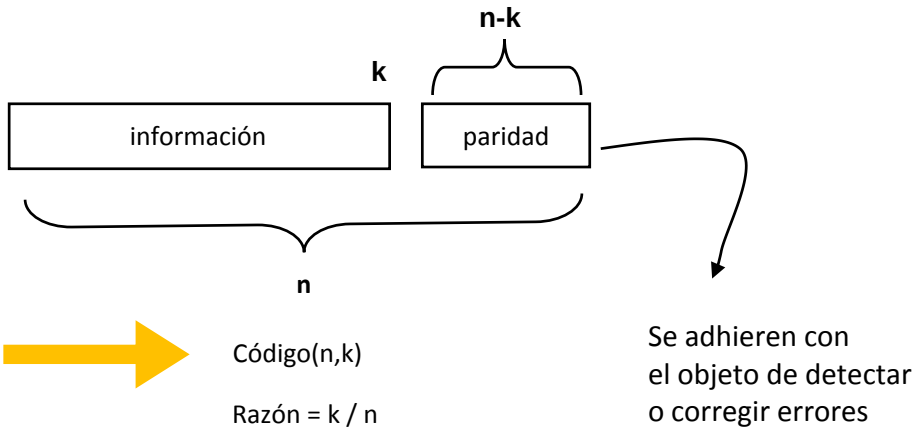

FIGURA 2.2: Códigos de bloque.

Algunos ejemplos son los códigos de *Hamming*, *Reed-Solomon* y *BCH*. A su vez, éstos se pueden clasificar en:

(a) *Códigos lineales:* Un código es lineal si cada combinación lineal de palabras válidas del código produce otra palabra válida (la suma módulo 2 de dos palabras código es también una palabra código).

(b) *Códigos cíclicos:* Un código es cíclico si es lineal y además, cada rotación cíclica de un código válido es también otro código válido.

2. **Códigos convolucionales:** Son aquellos en los que las palabras código (normalmente de longitud constante) dependen del mensaje actual y de un número determinado de los anteriores. Los códigos convolucionales son códigos lineales. El análisis de estos códigos es mucho más complejo que el de los códigos de bloque y se utilizan en canales con ruido blanco o gaussiano.

   Existen varios métodos de codificación de códigos convolucionales, aunque uno de los más usuales es el basado en registros de desplazamiento conectados con sumadores base 2 en los que se realiza la codificación. Por cada bit que entra en el codificador se obtienen n bits. El diagrama de bloques de uno de estos posibles codificadores convolucionales es el mostrado en la figura 2.3.

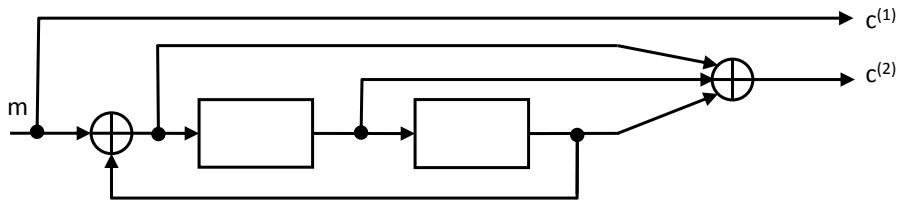

FIGURA 2.3: Codificador convolucional de Razón=$k/n = 1/2$.

La decodificación de un código convolucional consiste en escoger la secuencia más probable entre todas las posibles. Existen diversos algoritmos que permiten la decodificación de este tipo de códigos y la decodificación óptima se consigue mediante el algoritmo de Viterbi. El *algoritmo de Viterbi* realiza una decodificación de máxima verosimilitud encontrando la trayectoria a través del trellis con la mayor métrica (correlación máxima o distancia mínima) de la siguiente manera:

- Procesa la salida del demodulador de manera iterativa.
- En cada etapa en el trellis, compara la métrica de todas las trayectorias que ingresan a cada estado, conservando sólo aquellas trayectorias de mayor métrica, llamadas sobrevivientes, junto con su métrica.
- Procede por el trellis eliminando las trayectorias menos verosímiles.

3. **Turbo Códigos:** Técnicamente son unos códigos de bloque, pero funcionan como una combinación de un código de bloque y uno convolucional. Este tipo de códigos serán ampliamente explicados en el capítulo 3.

—3—
TURBO CÓDIGOS

EN ESTE CAPÍTULO SE PRESENTAN UN TIPO DE CÓDIGOS CORRECTORES DE ERRORES, LOS TURBO CÓDIGOS. ESTOS POSEEN UN GRAN INTERÉS POR SU ALTO RENDIMIENTO YA QUE PERMITEN TRANSMITIR A TASAS CERCANAS A LA CAPACIDAD DE SHANNON. AQUÍ SE EXPLICAN EN DETALLE SUS RESPECTIVOS ESQUEMAS DE TURBO CODIFICACIÓN Y DE TURBO DECODIFICACIÓN.

3.1 Introducción a los Turbo Códigos

Los Turbo Códigos son una clase de códigos propuestos en 1993 por Berrou, Glavieux y Thitimajashima que obtienen un excelente rendimiento en cuanto a tasa de error de bit (BER), proporcionando así comunicaciones fiables muy cerca del límite de Shannon (a 0.5 dB del límite).

La técnica de codificación de los turbo códigos se basa en la concatenación en paralelo de dos codificadores relativamente sencillos separados por un entrelazado. El conjunto es equivalente a un único codificador de memoria tan grande como la profundidad de entrelazado pero con un proceso de decodificación que en ningún caso alcanzará la complejidad del convolucional equivalente.

3.2 Esquema de Turbo Codificación

El turbo codificador más común consiste en la concatenación en paralelo de

dos codificadores convolucionales sistemáticos[1] y recursivos[2] (RSC) separados por un entrelazado pseudoaleatorio y unidos por un multiplexor y un bloque de perforación (éste último es opcional).

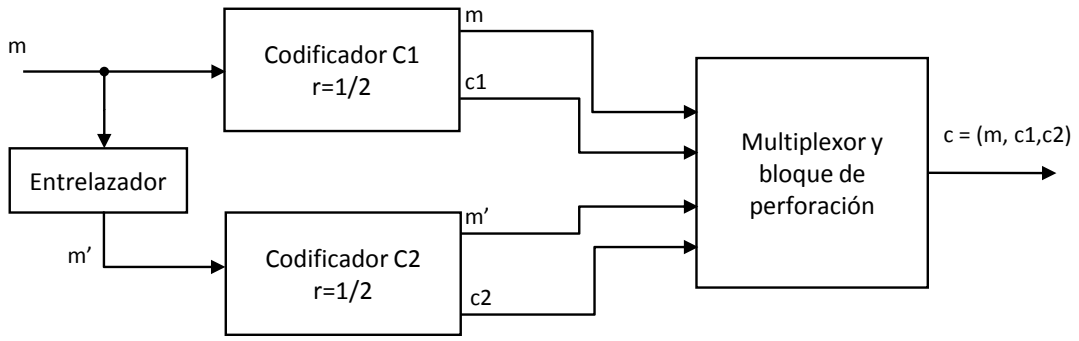

FIGURA 3.1: Turbo codificador.

El entrelazador permuta la información de entrada antes de ser codificada por el segundo codificador (C2) para obtener a la salida de cada codificador una versión incorrelada de la información. Esto significa que los bits de paridad a la salida de cada codificador son independientes y cuanto mejor sea el entrelazador utilizado, mayor será la independencia.

Un ejemplo de turbo codificador se puede ver en la figura 3.1. Los m bits de entrada están agrupados en secuencias de longitud igual al tamaño del entrelazador. La secuencia m' es el resultado del proceso de entrelazado. Los codificadores tienen una razón de r=$\frac{1}{2}$ (que es el caso habitual, pero existen otras combinaciones), por tanto, las longitudes de las secuencias de información (m y m') serán las mismas que las de paridad (c1 y c2). La secuencia obtenida a la salida del multiplexor será c=(m, c1, c2) (ya que el decodificador sabrá calcular m' a partir de m), así que la razón del turbo codificador es de r=$\frac{1}{3}$. Se pueden obtener razones superiores aplicando técnicas de perforación, es decir, eliminando bits de paridad después de codificar y en decodificación suponer que se ha enviado un determinado valor pero darle una fiabilidad muy baja (por ejemplo, si usamos una BPSK y, por tanto, se envía -1 o 1, en decodificación suponer que el valor recibido en el sitio perforado es un 0). De esta manera se consigue una razón r=$\frac{1}{2}$.

[1] Código sistemático: cuando a la salida del codificador los datos codificados se mantienen inalterados y se añade la información referente a la paridad al final.

[2] Código recursivo: aquel que realimenta la entrada con la salida del instante anterior.

Los elementos que condicionan el rendimiento del turbo código son:

- El entrelazador, especialmente su longitud y estructura.
- El número de elementos de memoria de cada uno de los códigos constituyentes.
- Los polinomios de los codificadores.
- Si los códigos están acabados o no (si el codificador acaba en un determinado estado o no).
- El patrón de perforación.
- La recursividad de los códigos constituyentes.

3.3 ESQUEMA DE TURBO DECODIFICACIÓN

LA TURBO CODIFICACIÓN toma su nombre de la realimentación que utiliza en la decodificación, como los motores turbo.

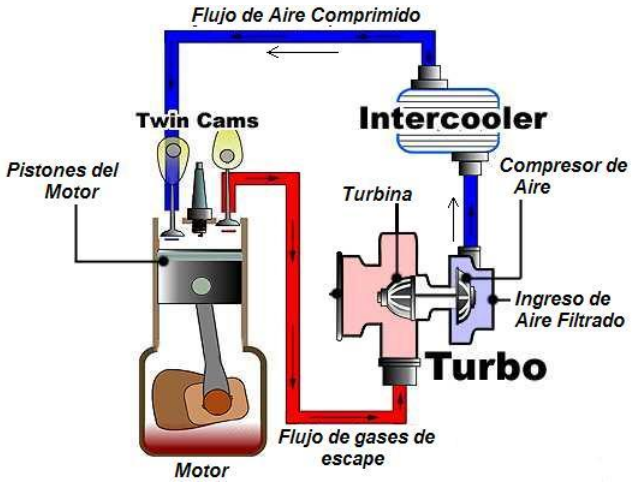

FIGURA 3.2: Esquema de un motor turbo.

Los turbo códigos emplean un proceso de decodificación iterativa soft en el que se usan diferentes versiones de la misma información (básicamente permutándola)

codificada con códigos relativamente sencillos con tal de obtener un código resultante con unas características muy cercanas al límite teórico de Shannon.

El turbo decodificador más común se muestra en la figura 3.3.

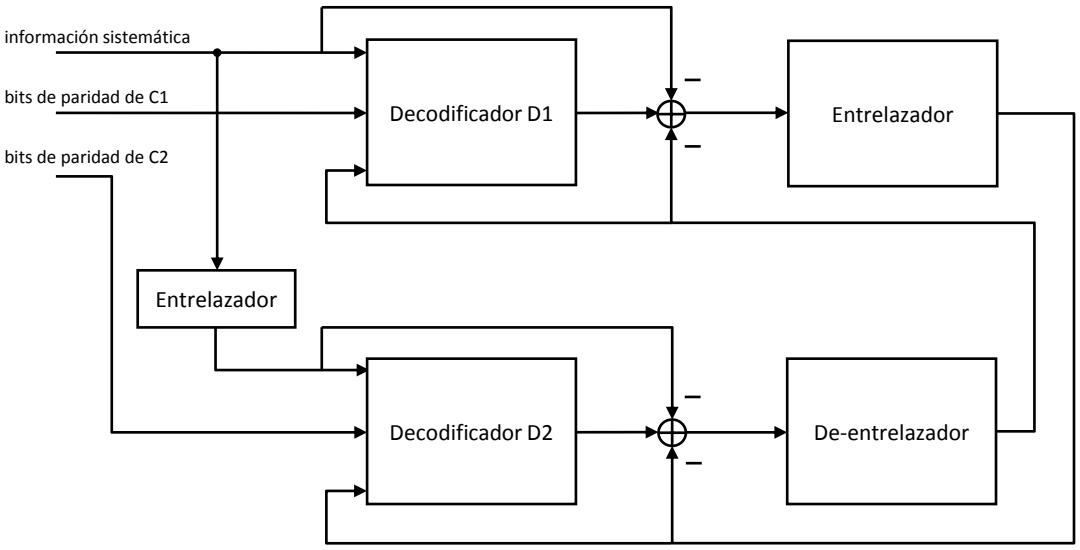

FIGURA 3.3: Turbo decodificador.

La decodificación iterativa empieza decodificando los códigos constituyentes individualmente, simultáneamente o no, a partir de la entrada recuperada del canal y de una determinada información $a\,priori$ que se calcula como la información extrínseca de la iteración previa (en el primer paso se supone que los bits pueden tomar los valores 0 o 1 de manera equiprobable, por tanto, esta información valdrá $\frac{1}{2}$). La información obtenida sobre los símbolos de la primera decodificación se comparte con los otros decodificadores y así repetidamente hasta que ya no hay mejora sobre el resultado (como media, por razones de complejidad, normalmente se usan entre 6 y 20 iteraciones). Es decir, la información obtenida del canal y la aportada por los otros decodificadores se utiliza para mejorar con cada iteración la decodificación de un determinado código constituyente y, por tanto, se mejora la decodificación conjunta.

Otro responsable del aumento de rendimiento que tiene este tipo de código es el uso de la denominada información 'soft' en lugar de las decisiones 'hard'. En otras palabras, el uso del valor de la estimación de un símbolo en lugar de usar los símbolos '0' y '1'. La información soft se evalúa de forma logarítimica usando

la razón de verosimilitud logarítmica (LLR, por sus siglas en inglés Log-Likelihood Ratios). La LLR para un bit b_i se define como el logaritmo del cociente entre las probabilidades de que el bit decodificado sea '0' o '1'.

$$L(b_i) = \ln\left(\frac{P(b_i = +1)}{P(b_i = -1)}\right) \tag{3.1}$$

Esta medida es muy adecuada porque su resultado es un número con signo. Su signo indica directamente si el bit estimado es un '0' (signo negativo) o un '1' (signo positivo), mientras que su magnitud da una medida cuantitativa de la probabilidad de que el bit decodificado sea un '0' o un '1'. Por este motivo es más conveniente definir la LLR como la probabilidad de que el bit recibido sea -1 o +1, usando el alfabeto polar en vez del binario $\{0, 1\}$.

Por tanto, el decodificador no sólo aporta información sobre el valor de cada bit, sino que además informa de la veracidad de éste.

Los algoritmos más usados para optimizar la probabilidad de error de bit de cada código constituyente son el algoritmo de Viterbi con salida soft (SOVA) o el criterio de máxima probabilidad *a posteriori* (MAP).

El algoritmo MAP es óptimo en cuanto a la minimización de la BER decodificada cuando se usa para decodificar códigos convolucionales. El algoritmo de Viterbi, también usado para decodificar estos códigos, minimiza la probabilidad de que un camino incorrecto a través del trellis sea seleccionado por el decodificador. Es decir, a diferencia del algoritmo MAP, el algoritmo de Viterbi permite hacer de manera óptima la búsqueda del camino en el trellis más parecido a la secuencia recibida sin evaluar todas las posibles distancias.

3.4 EL ALGORITMO MAP

EL ALGORITMO DE MÁXIMA PROBABILIDAD *a posteriori* (MAP), conocido también como algoritmo BCJR por las iniciales de sus autores (Bahl Cocke, Jelinek y Raviv), fue formalmente presentado en 1974 por Bahl *et ál.* (2) como una alternativa para la decodificación de códigos convolucionales. La decodificación con algoritmos MAP ha tenido un resurgimiento desde el descubrimiento de los codificadores turbo en 1993. Los decodificadores MAP realizan una decisión símbolo a símbolo óptima, y además proveen salidas soft, que consisten en una versión cuantificada de la decisión del decodificador que puede ser vista como la probabilidad de dicha decisión,

características necesarias en sistemas de decodificación concatenada como los turbo códigos.

El rendimiento del algoritmo MAP, en muchos sistemas, es similar al del algoritmo de Viterbi. Sin embargo, la complejidad del algoritmo MAP es mayor, por eso no fue ampliamente usado hasta el descubrimiento de los turbo códigos.

3.4.1 INTRODUCCIÓN MATEMÁTICA

El problema en la decodificación de turbo códigos es esencialmente determinar las estimaciones MAP o decisiones soft de los estados y las transiciones del codificador trellis, visto éste como un modelo oculto de Markov[3] cuya secuencia de salida es observada a través de un canal discreto sin memoria. Esto se muestra en la figura 3.4.

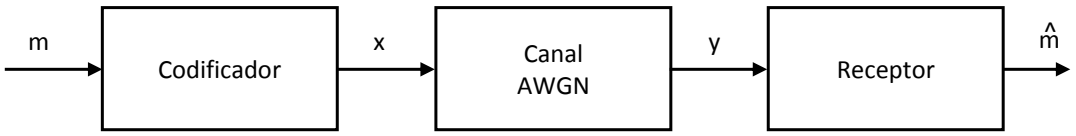

FIGURA 3.4: Escenario para el algoritmo MAP.

El modelo oculto de Markov representa el codificador trellis con un número finito de estados u=0, 1, 2...U-1. El estado a su entrada en el momento i, cuya notación es S_i, tiene como salida X_i. La secuencia de estados desde el instante i hasta el j $S_i^j = \{S_i, S_{i+1}...S_j\}$ tendrá como correspondiente salida la secuencia $X_i^j = \{X_i, X_{i+1}...X_j\}$. Las transiciones de estado están determinadas por las probabilidades de transición

$$p_i(u/u') = P(S_i = u/S_{i-1} = u') \tag{3.2}$$

y las correspondientes probabilidades a la salida

$$q_i(X/\{u', u\}) = P(X_i = x/\{S_{i-1} = u', S_i = u\}) \tag{3.3}$$

El modelo oculto de Markov genera una secuencia X_1^n que empieza en el estado S_0 y acaba en el mismo estado S_0. La salida X_1^n es la entrada de un canal discreto

[3] Modelo oculto de Markov: Un modelo oculto de Markov puede considerarse como un caja negra donde la secuencia de símbolos de salida generados a lo largo del tiempo es visible, pero la secuencia de estados por los que se ha pasado para generar la anterior serie se desconoce.

ruidoso y sin memoria que genera la secuencia distorsionada $Y_1^n = \{Y_1, Y_2...Y_n\}$. Las probabilidades de transición en este tipo de canal se definen para cada instante $1 \leq i \leq n$,

$$P(Y_1^i/X_1^i) = \prod_{j=1}^{i} R(Y_j/X_j) \tag{3.4}$$

El término $R(Y_j/X_j)$ determina la probabilidad de que en el instante j, el símbolo Y_j sea la salida del canal si el símbolo X_j ha sido la entrada al canal.

Un decodificador para este proceso de Markov tiene que estimar la probabilidad MAP de los estados y salidas del modelo oculto de Markov observando la secuencia de salida $Y_1^n = \mathbf{Y} = \{Y_1, Y_2...Y_n\}$.

3.4.2 CÁLCULO DE LA LLR

El algoritmo MAP da, para cada bit decodificado (b_i), la probabilidad de que el bit sea +1 o -1, dada la secuencia recibida **Y**. Esto es equivalente a encontrar la LLR *a posteriori* $L(b_i \mid \mathbf{Y})$, donde

$$L(b_i \mid \mathbf{Y}) = \ln\left(\frac{P(b_i = +1 \mid \mathbf{Y})}{P(b_i = -1 \mid \mathbf{Y})}\right) \tag{3.5}$$

En un trellis, si el estado anterior $S_{i-1} = u'$ y el presente $S_i = u$ son conocidos, el bit de entrada b_i que ha causado la transición entre estados será conocido. El hecho de que las transiciones entre el anterior estado $S_{i-1} = u'$ y el presente $S_i = u$ en un trellis sean exclusivas (es decir, sólo una de ellas puede haber ocurrido en el codificador), nos permite escribir (3.5) como

$$L(b_i \mid \mathbf{Y}) = L(b_i \mid Y_1^n) = \ln\left(\frac{\Sigma_{\{u',u\}\Rightarrow b_i=+1} P(S_{i-1} = u', S_i = u, Y_1^n)}{\Sigma_{\{u',u\}\Rightarrow b_i=-1} P(S_{i-1} = u', S_i = u, Y_1^n)}\right) \tag{3.6}$$

donde $\{u', u\} \Rightarrow b_i = +1$ es el conjunto de transiciones que van desde el estado previo $S_{i-1} = u'$ hasta el presente $S_i = u$ y que pueden ocurrir cuando el bit de entrada sea $b_i = +1$. Lo mismo ocurre con $\{u', u\} \Rightarrow b_i = -1$ para el bit de entrada $b_i = -1$.

Ahora vamos a considerar las probabilidades individuales del numerador y denominador de (3.6). La secuencia recibida Y_1^n puede dividirse en tres partes:

- Y_1^{i-1} : la secuencia recibida antes de la presente transición.

- Y_i : el símbolo asociado a la presente transición.
- Y_{i+1}^n : la secuencia recibida después de la presente transición.

Podemos definir la probabilidad $P(S_{i-1} = u', S_i = u, Y_1^n)$ como

$$\sigma_i(u', u) = P(S_{i-1} = u', S_i = u, Y_1^n) = P(S_{i-1} = u', S_i = u, Y_1^{i-1}, Y_i, Y_{i+1}^n) \quad (3.7)$$

Teniendo en cuenta que en un canal sin memoria la futura secuencia recibida Y_{i+1}^n dependerá únicamente del estado presente S_i y no del estado anterior S_{i-1} o de la presente Y_i y pasada Y_1^{i-1} secuencia, y el teorema de Bayes:

$$P(A, B) = P(A \mid B)P(B) = P(B \mid A)P(A)$$

$\sigma_i(u', u)$ puede reescribirse como

$$\begin{aligned}
\sigma_i(u', u) &= P(Y_{i+1}^n \mid S_{i-1} = u', S_i = u, Y_1^{i-1}, Y_i)P(S_{i-1} = u', S_i = u, Y_1^{i-1}, Y_i) \\
&= P(Y_{i+1}^n \mid S_i = u)P(S_{i-1} = u', S_i = u, Y_1^{i-1}, Y_i) \\
&= P(Y_{i+1}^n \mid S_i = u)P(S_i = u, Y_i \mid S_{i-1} = u', Y_1^{i-1})P(S_{i-1} = u', Y_1^{i-1}) \\
&= P(Y_{i+1}^n \mid S_i = u)P(S_i = u, Y_i \mid S_{i-1} = u')P(S_{i-1} = u', Y_1^{i-1}) \quad (3.8)
\end{aligned}$$

Por tanto

$$\begin{aligned}
\sigma_i(u', u) &= P(S_{i-1} = u', Y_1^{i-1})P(S_i = u, Y_i \mid S_{i-1} = u')P(Y_{i+1}^n \mid S_i = u) \\
&= \alpha_{i-1}(u')\gamma_i(u', u)\beta_i(u) \quad (3.9)
\end{aligned}$$

donde

$$\alpha_{i-1}(u') = P(S_{i-1} = u', Y_1^{i-1}) \quad (3.10)$$

es la probalidad de que el trellis esté en el estado u' en el momento $i-1$ y de que la secuencia recibida hasta este momento sea Y_1^{i-1} (probabilidad asociada al pasado de la secuencia $\mathbf{Y}$)

$$\beta_i(u) = P(Y_{i+1}^n \mid S_i = u) \quad (3.11)$$

es la probalidad de que la secuencia recibida sea Y_{i+1}^n si el trellis está en el estado u en el momento i (probabilidad asociada al futuro de la secuencia $\mathbf{Y}$). Y finalmente

$$\gamma_i(u', u) = P(S_i = u, Y_i \mid S_{i-1} = u') \quad (3.12)$$

es la probalidad de que el trellis estuviera en el estado u' en el momento $i-1$, haya una transición hacia el estado u y la secuencia recibida para dicha transición sea

Y_i (probabilidad asociada al presente de la secuencia $\mathbf{Y}$). Ahora podemos escribir la LLR condicional de b_i dada la secuencia Y_1^n como

$$\begin{aligned} L(b_i \mid Y_1^n) &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1} P(S_{i-1}=u', S_i=u, Y_1^n)}{\sum_{\{u',u\}\Rightarrow b_i=-1} P(S_{i-1}=u', S_i=u, Y_1^n)}\right) \\ &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1} \alpha_{i-1}(u')\gamma_i(u',u)\beta_i(u)}{\sum_{\{u',u\}\Rightarrow b_i=-1} \alpha_{i-1}(u')\gamma_i(u',u)\beta_i(u)}\right) \end{aligned} \tag{3.13}$$

El algoritmo MAP encuentra $\alpha_i(u)$ y $\beta_i(u)$ para todos los u estados posibles a través del trellis (es decir, para u=0, 1...U-1) y $\gamma_i(u',u)$ para todas las posibles transiciones desde el estado $S_{i-1}=u'$ hasta el estado $S_i=u$ para todos los posibles estados (es decir, para u=0, 1,...U-1). Estos valores se usan en la expresión (3.13) para calcular la LLR condicional $L(b_i \mid Y_1^n)$ que entrega el decodificador MAP.

A continuación describimos cómo pueden ser calculados los valores de $\alpha_i(u)$, $\beta_i(u)$ y $\gamma_i(u',u)$.

3.4.3 CÁLCULO RECURSIVO DE LOS VALORES $\alpha_i(u)$

Partiendo de la expresión (3.10), usando el teorema de Bayes y asumiendo que el canal no tiene memoria, podemos definir $\alpha_i(u)$ como

$$\begin{aligned} \alpha_i(u) &= P(S_i=u, Y_1^i) = P(S_i=u, Y_1^{i-1}, Y_i) \\ &= \sum_{u'=0}^{U-1} P(S_{i-1}=u', S_i=u, Y_1^{i-1}, Y_i) \\ &= \sum_{u'=0}^{U-1} P(S_{i-1}=u', Y_1^{i-1}) P(S_i=u, Y_i/\{S_{i-1}=u', Y_1^{i-1}\}) \\ &= \sum_{u'=0}^{U-1} P(S_{i-1}=u', Y_1^{i-1}) P(S_i=u, Y_i/S_{i-1}=u') \\ &= \sum_{u'=0}^{U-1} \alpha_{i-1}(u')\gamma_i(u',u) \end{aligned} \tag{3.14}$$

Asumiendo que el trellis tiene el estado inicial $S_0=0$, las condiciones iniciales que usa el decodificador son

$$\begin{aligned} \alpha_0(S_0=0) &= 1 \\ \alpha_0(S_0=u) &= 0 \quad si\ u\neq 0 \end{aligned} \tag{3.15}$$

Y ocurre lo mismo para i=1, 2...,n-1.

Por tanto, una vez conocidos los valores $\gamma_i(u', u)$, los valores de $\alpha_i(u)$ pueden ser calculados de forma recursiva desde el inicio del trellis hasta el final (sentido $forward$).

3.4.4 CÁLCULO RECURSIVO DE LOS VALORES $\beta_i(u)$

Los valores de $\beta_i(u)$ también se calculan de manera recursiva. Utilizando una derivación similar a la usada en (3.14) se pueden obtener dichos valores

$$\begin{aligned}
\beta_i(u) &= P(Y_{i+1}^n \mid S_i = u) \\
&= \sum_{u'=0}^{U-1} P(\{S_{i+1} = u', Y_{i+1}^n\} \mid S_i = u) \\
&= \sum_{u'=0}^{U-1} P(\{S_{i+1} = u', Y_{i+1}\} \mid S_i = u) P(Y_{i+2}^n \mid S_{i+1} = u') \\
&= \sum_{u'=0}^{U-1} \beta_{i+1}(u') \gamma_{i+1}(u, u') \qquad (3.16)
\end{aligned}$$

Cuando se trabaja con un trellis acabado, es decir, que empieza y acaba en el estado cero S_0, las condiciones iniciales que usa el decodificador son

$$\begin{aligned}
\beta_n(S_0 = 0) &= 1 \\
\beta_n(S_0 = u) &= 0 \quad si \;\; u \neq 0 \qquad (3.17)
\end{aligned}$$

Si no es el caso, entonces $\beta_n(S_0 = u) = 1$ para todo valor de u.

Por tanto, una vez conocidos los valores $\gamma_i(u, u')$, los valores de $\beta_i(u)$ pueden ser calculados de forma recursiva desde el final del trellis hasta el inicio (sentido $backward$).

3.4.5 CÁLCULO DE LOS VALORES $\gamma_i(u', u)$

Los coeficientes $\alpha_{i-1}(u')$ y $\beta_i(u)$ se calculan de manera recursiva en función de los coeficientes $\gamma_i(u', u)$, por tanto dichos coeficientes deben ser calculados previamente para poder obtener todos los valores necesarios en el algoritmo MAP. Haciendo uso del teorema de Bayes y las propiedades de un canal sin memoria, puede escribirse $\gamma_i(u', u)$ como

$$\begin{aligned}\gamma_i(u',u) &= P(S_i=u, Y_i/S_{i-1}=u') \\ &= P(Y_i/\{u',u\})P(u',u) \\ &= P(Y_i/\{u',u\})P(b_i) \end{aligned} \tag{3.18}$$

La probabilidad de bit en el instante i puede ser calculada en función de la LLR haciendo las siguientes operaciones

$$L(b_i) = \ln\left(\frac{P(b_i=+1)}{P(b_i=-1)}\right)$$

$$e^{L(b_i)} = \frac{P(b_i=+1)}{P(b_i=-1)} = \frac{P(b_i=+1)}{1-P(b_i=+1)}$$

$$P(b_i=+1) = \frac{e^{L(b_i)}}{1+e^{L(b_i)}} = \frac{1}{1+e^{-L(b_i)}}$$

$$P(b_i=-1) = \frac{e^{-L(b_i)}}{1+e^{-L(b_i)}} = \frac{1}{1+e^{+L(b_i)}}$$

$$P(b_i=\pm 1) = \frac{e^{-L(b_i)/2}}{1+e^{-L(b_i)}}e^{\pm L(b_i)/2} = \frac{e^{-L(b_i)/2}}{1+e^{-L(b_i)}}e^{(b_iL(b_i))/2} = C_1 e^{(b_iL(b_i))/2} \tag{3.19}$$

Por otro lado, el cálculo del término $P(Y_i/\{u',u\})$ es equivalente a calcular la probabilidad $P(Y_i \mid X_i)$, donde X_i es el vector asociado a la transición desde $S_{i-1}=u'$ hasta $S_i = u$, que es en general un vector de n bits. Si el canal es un canal sin memoria, entonces esta probabilidad es

$$P(Y_i/\{u',u\}) = P(Y_i/X_i) = \prod_{k=1}^{n} P(y_{ik}/x_{ik}) \tag{3.20}$$

donde y_{ik} y x_{ik} son los bits de los vectores recibidos y transmitidos (Y_i y X_i). Si la transmisión se hace en formato polar a través de un canal AWGN, los bits transmitidos x_{ik} adquieren los valores +1 y -1, y

$$\begin{aligned} P(Y_i/\{u',u\}) &= \prod_{k=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-E_b}{2\sigma^2}\sum_{k=1}^{n}(y_{ik}-xik)^2} \\ &= \frac{1}{(\sqrt{2\pi}\sigma)^n} e^{\frac{-E_b}{2\sigma^2}\sum_{k=1}^{n}(y_{ik}^2+x_{ik}^2)} e^{\frac{E_b}{\sigma^2}\sum_{k=1}^{n}(y_{ik}x_{ik})} \\ &= C_2 e^{\frac{E_b}{\sigma^2}\sum_{k=1}^{n}(y_{ik}x_{ik})} \end{aligned} \tag{3.21}$$

donde tan sólo el término $e^{\frac{E_b}{\sigma^2}\sum_{k=1}^{n}(y_{ik}x_{ik})}$ es significante porque el resto de los términos de la expresión son constantes. La expresión para calcular los coeficientes $\gamma_i(u', u)$ es

$$\gamma_i(u', u) = Ce^{(b_i L(b_i))/2} e^{\frac{E_b}{\sigma^2}\sum_{k=1}^{n}(y_{ik}x_{ik})} \quad donde\ C = C_1 C_2 \tag{3.22}$$

Estos coeficientes pueden ser calculados teniendo en cuenta la información $a\,priori$ de cada decodificador constituyente $L(b_i)$ (que es la información extrínseca obtenida del otro decodificador, es decir, es la información obtenida en la iteración previa) y la información de canal L_c. Para obtener la relación entre la información de canal y algunos de los valores de la expresión (3.22) se usa la LLR condicional para un canal AWGN cuando se recibe la señal y_i

$$P(y_i/x_i = +1) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-E_b}{2\sigma^2}(y_i-1)^2}$$

$$P(y_i/x_i = -1) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-E_b}{2\sigma^2}(y_i+1)^2}$$

$$\begin{aligned} L(y_i/x_i) &= \ln\left(\frac{P(y_i/x_i=+1)}{P(y_i/x_i=-1)}\right) = \ln\left(\frac{e^{\frac{-E_b}{2\sigma^2}(y_i-1)^2}}{e^{\frac{E_b}{2\sigma^2}(y_i+1)^2}}\right) \\ &= \frac{-E_b}{2\sigma^2}(y_i-1)^2 + \frac{E_b}{2\sigma^2}(y_i+1)^2 = 2\frac{E_b}{\sigma^2}y_i = L_c y_i \end{aligned} \tag{3.23}$$

Ahora puede reescribirse la ecuación (3.22) para los coeficientes $\gamma_i(u', u)$ en función de la constante de proporción $L_c = 2\frac{E_b}{\sigma^2}$, que es una medida de la relación señal/ruido del canal.
Si además se tiene en cuenta que las estructuras de turbo códigos más comunes están formadas por codificadores RSC de razón r=$\frac{1}{2}$ (es decir, que la asignación para la entrada de las transiciones trellis es hecha para un único bit), se puede distinguir entre la información del mensaje (b_i) y la información redundante y se obtiene

$$\begin{aligned} \gamma_i(u', u) &= Ce^{(b_i L(b_i))/2} e^{\frac{L_c}{2}\sum_{k=1}^{n}(y_{ik}x_{ik})} \\ &= Ce^{(b_i L(b_i))/2} e^{\frac{L_c}{2}y_{i1}x_{i1}} e^{\frac{L_c}{2}\sum_{k=2}^{n}(y_{ik}x_{ik})} \\ &= Ce^{(b_i L(b_i))/2} e^{\frac{L_c}{2}y_{i1}b_i} e^{\frac{L_c}{2}\sum_{k=2}^{n}(y_{ik}x_{ik})} \\ &= Ce^{(b_i L(b_i))/2} e^{\frac{L_c}{2}y_{i1}b_i} \gamma_{i\_extr}(u', u) \\ &= Ce^{0.5(L(b_i)b_i + L_c y_{i1} b_i)} \gamma_{i\_extr}(u', u) \end{aligned} \tag{3.24}$$

3.4.6 RESUMEN DEL ALGORITMO MAP

Resumiendo, la decodificación MAP de una secuencia recibida Y_1^n para dar una LLR $a\,posteriori\ L(b_i \mid Y_1^n)$ se calcula de la siguiente manera:

1. Se establecen las condiciones iniciales $\alpha_0(0) = 1$ y $\alpha_0(u) = 0$, $u \neq 0$, y las condiciones de contorno $\beta_n(0) = 1$ y $\beta_n(u) = 1$, $u \neq 0$ para $u = 0, 1, 2...U-1$.

2. Después de recibir Y_i, el decodificador calcula $\gamma_i(u', u)$ con la ecuación (3.24) y determina $\alpha_i(u)$ con la ecuación (3.14). Los valores son almacenados para cada i y cada u.

3. Después de recibir la secuencia Y_i^n entera, el decodificador calcula de manera recursiva los valores de $\beta_i(u)$ usando la expresión (3.16).

4. Una vez obtenidos los valores de $\gamma_i(u', u)$, $\alpha_i(u)$ y $\beta_i(u)$ ya se puede obtener el valor de la LLR, $L(b_i \mid Y_1^n)$. Teniendo en cuenta que en la definición de la LLR, vista en (3.13), el numerador está compuesto por términos asociados con $b_i = +1$, mientras que el denominador lo está por términos asociados con $b_i = -1$, $L(b_i \mid Y_1^n)$ puede escribirse como

$$\begin{aligned} L(b_i \mid Y_1^n) &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1}\ \alpha_{i-1}(u')\gamma_i(u',u)\beta_i(u)}{\sum_{\{u',u\}\Rightarrow b_i=-1}\ \alpha_{i-1}(u')\gamma_i(u',u)\beta_i(u)}\right) \\ &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1}\ \alpha_{i-1}(u')e^{+L(b_i)/2}e^{+L_c y_{i1}/2}\gamma_{i\_extr}(u',u)\beta_i(u)}{\sum_{\{u',u\}\Rightarrow b_i=-1}\ \alpha_{i-1}(u')e^{-L(b_i)/2}e^{-L_c y_{i1}/2}\gamma_{i\_extr}(u',u)\beta_i(u)}\right) \\ &= L(b_i) + L_c y_{i1} + L_e(b_i) \end{aligned} \tag{3.25}$$

donde

$$L_e(b_i) = \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1}\ \alpha_{i-1}(u')\gamma_{i\_extr}(u',u)\beta_i(u)}{\sum_{\{u',u\}\Rightarrow b_i=-1}\ \alpha_{i-1}(u')\gamma_{i\_extr}(u',u)\beta_i(u)}\right) \tag{3.26}$$

es la llamada LLR extrínseca, que es la estimación o la decisión soft que cada decodificador comunica al otro con respecto al bit de mensaje b_i. Reagrupando los términos de la expresión (3.25) dicho valor puede reescribirse como

$$L_e(b_i) = L(b_i \mid Y_1^n) - L(b_i) - L_c y_{i1} \tag{3.27}$$

3.5 TURBO DECODIFICACIÓN

EL ALGORITMO DE DECODIFICACIÓN MAP es un algoritmo de decodificación iterativo donde cada decodificador constituyente genera decisiones soft o estimaciones de los bits del mensaje transmitido. Dichas estimaciones son calculadas usando la información de canal de la secuencia recibida y la información $a\,priori$ suministrada por el otro decodificador en la iteración previa. Esta información transmitida entre los decodificadores es la LLR extrínseca que cada uno de ellos determina en la iteración previa y que se convierte en la información $a\,priori$ del otro decodificador en la iteración presente.

Este proceso iterativo de intercambio de información es tal que, bajo ciertas condiciones, las estimaciones de los bits del mensaje transmitido son más cercanas a los verdaderos valores cuando el número de iteraciones incrementa.

La información $a\,priori$ de un bit es información conocida antes del comienzo de la decodificación que no tiene relación con la secuencia recibida, ni con la información de codificación (aquella proporcionada por la estructura trellis del código). En la primera iteración, el primer decodificador no tiene información $a\,priori$ de los bits, por tanto, todos los bits que forman el mensaje son equiprobables. Esto significa que la LLR $a\,priori$ $L(b_i)$ del primer decodificador en la primera iteración es igual a cero ya que

$$L_1^{(1)}(b_i) = \ln\left(\frac{P(b_i=+1)}{P(b_i=-1)}\right) = \ln\left(\frac{0.5}{0.5}\right) = 0$$

El primer decodificador tiene en cuenta esta información $a\,priori$ y la información de canal, que está proporcionada por los valores de la secuencia recibida afectados por el factor de canal L_c. Esta secuencia $L_c\mathbf{Y}_1^{(1)}$ consiste en la información sistemática y en los bits de paridad creados por el codificador C1.

El primer decodificador utiliza su información $a\,priori$ y su información de canal para determinar la primera estimación de la LLR $L_1^{(1)}(b_i/\mathbf{Y})$. El subíndice identifica el decodificador que ha generado la LLR, y el superíndice identifica el número de la iteración. Para este cálculo el decodificador necesita determinar los coeficientes $\gamma_i(u', u)$ y después obtener los valores de $\alpha_{i-1}(u')$ y $\beta_i(u)$ para así, finalmente, poder determinar la LLR $L_1^{(1)}(b_i/\mathbf{Y})$. Una vez obtenidas estas estimaciones, el decodificador debe comunicar al otro decodificador la información extrínseca. La información extrínseca es aquella que no incluye ni la información $a\,priori$ utilizada en el cálculo actual de $L_1^{(1)}(b_i/\mathbf{Y})$, ni la información de canal de los bits del mensaje para el que se calcula la información extrínseca. La información extrínseca $L_{e1}^{(1)}(b_i)$

se calcula mediante la expresión (3.27).

El segundo decodificador es entonces capaz de realizar sus estimaciones con la información disponible. Este decodificador usa la secuencia recibida $L_c\mathbf{Y_2^{(1)}}$ que contiene las muestras de los bits de información sistemática entrelazadas, y las muestras de los correspondientes bits de paridad generados por el codificador C_2 sobre la secuencia entrelazada de los bits de información sistemática. El segundo decodificador toma como su información $a\,priori$ la información extrínseca $L_{e1}^{(1)}(b_i)$ de cada bit de mensaje b_i, generada en la iteración actual por el primer decodificador. Sin embargo, tal y como se ha aplicado el entrelazado a los bits del mensaje, esta información extrínseca también debe ser reordenada de acuerdo con las normas de entrelazado antes de ser procesada por el segundo decodificador. Si se define la operación realizada por el entrelazador como $I\{\cdot\}$, entonces $L_2^{(1)}(b_i) = I\{L_{e1}^{(1)}(b_i)\}$. El segundo decodificador toma $L_2^{(1)}(b_i)$ como su información $a\,priori$ y, junto a la información de canal $L_c\mathbf{Y_2^{(1)}}$, es capaz de determinar la LLR $L_2^{(1)}(b_i/\mathbf{Y})$. Finalmente, usando la ecuación (3.27) se calcula la información extrínseca $L_{e2}^{(1)}(b_i)$ que será comunicada al primer decodificador.

Como la información extrínseca $L_{e2}^{(1)}(b_i)$ correspondiente al bit de mensaje b_i está afectada por el entrelazador, ya que el segundo decodificador recibe la versión entrelazada, se lleva a cabo el de-entrelazado para reordenar esta información antes de ser transmitida al primer decodificador. La operación de de-entrelazado se define con el operador $I^{-1}\{\cdot\}$. La información extrínseca proporcionada por el segundo decodificador se reordena para ser convertida en información $a\,priori$ del primer decodificador, por tanto $L_1^{(2)}(b_i) = I^{-1}\{L_{e2}^{(1)}(b_i)\}$. En esta segunda iteración el primer decodificador utiliza de nuevo la misma información de canal disponible $L_c\mathbf{Y}_1^{(1)}$, pero ahora la información $a\,priori$ es diferente de cero, porque esta información está actualizada gracias a la información extrínseca proporcionada por el segundo decodificador en la primera iteración. De esta manera el primer decodificador produce estimaciones mejoradas o LLRs de los bits de mensaje $L_1^{(2)}(b_i/\mathbf{Y})$.

Este proceso iterativo continúa, y con cada iteración la media de la BER de los bits decodificados disminuye. Sin embargo, la mejora del rendimiento por cada iteración llevada a cabo disminuye cuando el número de iteraciones aumenta. Por lo tanto, normalmente sólo se realizan entre 6 y 20 iteraciones ya que, normalmente, a partir de 8 iteraciones no hay una mejora importante en el rendimiento obtenido.

La figura 3.5 describe este procedimiento de decodificación iterativa de los turbo códigos.

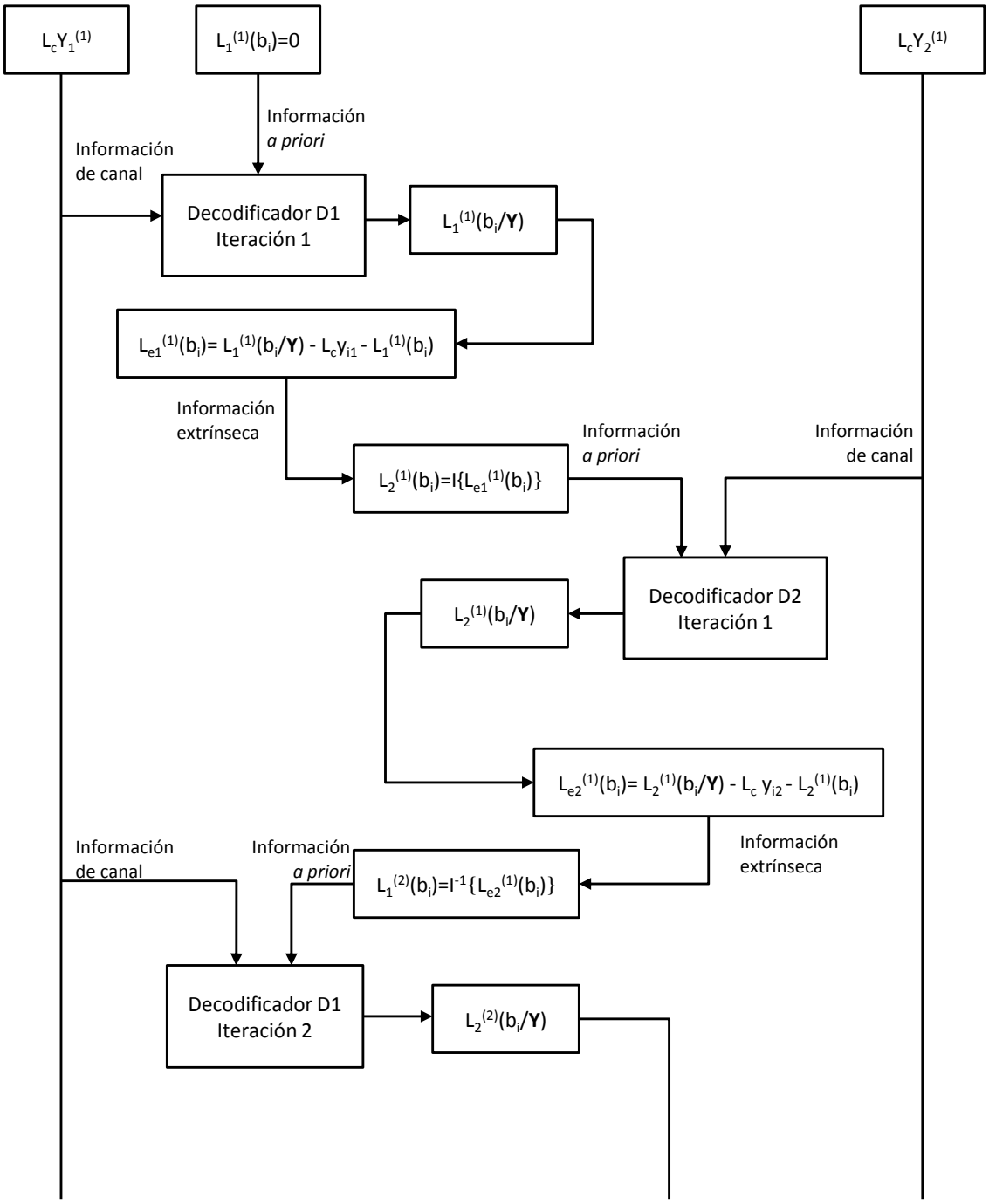

FIGURA 3.5: Decodificación iterativa de los turbo códigos.

3.6 MODIFICACIONES DEL ALGORITMO MAP

EL ALGORITMO MAP TIENE UNA GRAN COMPLEJIDAD ya que debe efectuar una enorme cantidad de multiplicaciones y sumas por cada estimación que produce de la probabilidad $a\,posteriori$ de cada bit de información decodificado. Este problema se puede aminorar si el algoritmo entero se lleva a cabo en el dominio logarítmico, en vez de esperar hasta el último paso para tomar el logaritmo de la LLR. El principal beneficio de ejecutar el algoritmo en el dominio logarítmico consiste en que la multiplicación se convierte en suma. Esta es la idea principal en la que se basan los algoritmos Max-Log-MAP (propuesto por W. Koch y A. Baier (5) y Erfanian *et ál.* (3)) y Log-MAP (propuesto por P. Robertson *et ál.* en 1995 (14)) para reducir la complejidad del algoritmo MAP.
La diferencia entre los algoritmos Max-Log-MAP y Log-MAP radica principalmente en la forma en que efectúan una operación de suma en el dominio logarítmico. El desempeño del algoritmo Max-Log-MAP en comparación con el del algoritmo MAP es subóptimo debido a que utiliza una aproximación para efectuar las operaciones de suma en el dominio logarítmico. El algoritmo Log-MAP fue propuesto para corregir dicha aproximación y su desempeño es idéntico al del algoritmo MAP. Sin embargo, tanto el algoritmo Log-MAP como el Max-Log-MAP reducen en buena medida la complejidad del algoritmo MAP.

3.6.1 EL ALGORITMO MAX-LOG-MAP

El algoritmo MAP calcula las LLRs $a\,posteriori\ L(b_i \mid Y_1^n)$ usando (3.13). Para realizar dicho cálculo se deben obtener los valores de $\alpha_i(u)$, $\beta_i(u)$ y $\gamma_i(u', u)$ usando las expresiones (3.14), (3.16) y (3.24) respectivamente.
El algoritmo Max-Log-MAP simplifica estas ecuaciones en el dominio del logaritmo aritmético utilizando la aproximación

$$\ln(\sum_j e^{a_j}) \approx \max_j(a_j) \tag{3.28}$$

donde $\max_j(a_j)$ significa el máximo valor de a_j. Definiendo $A_i(u)$, $B_i(u)$ y $\Gamma_i(u', u)$ de la siguiente manera:

$$A_i(u) \triangleq \ln(\alpha_i(u)) \tag{3.29}$$

$$B_i(u) \triangleq \ln(\beta_i(u)) \tag{3.30}$$

$$\Gamma_i(u', u) \triangleq \ln(\gamma_i(u', u)) \tag{3.31}$$

Se puede reescribir (3.14) como

$$\begin{aligned} A_i(u) &\triangleq \ln(\alpha_i(u)) \\ &= \ln(\sum_{u'=0}^{U-1} \alpha_{i-1}(u')\gamma_i(u', u)) \\ &= \ln(\sum_{u'=0}^{U-1} e^{\ln[\alpha_{i-1}(u')\gamma_i(u', u)]}) \\ &\approx \max_{u'}(\ln[\alpha_{i-1}(u')] + \ln[\gamma_i(u', u)]) \\ &= \max_{u'}(A_{i-1}(u') + \Gamma_i(u', u)) \end{aligned} \tag{3.32}$$

Debido al uso de la aproximación (3.28), el valor de $A_i(u)$ en el algoritmo Max-Log-MAP da la probabilidad del camino más probable a través del trellis que va hacia el estado u en lugar de la probabilidad de todos los caminos a través del trellis que alcanzan el estado u. Esta aproximación es una de las razones por las que el algoritmo Max-log-MAP tiene un rendimiento subóptimo comparado al algoritmo MAP. Se debe tener en cuenta que para trellis binarios la suma y la maximización sobre los previos estados u' que llegan al estado u llevada a cabo en (3.32) serán únicamente aplicadas a dos, ya que sólo habrá dos estados previos u' con camino hacia al estado presente u. Para todos los otros valores de u' se tendrá $\gamma_i(u', u) = 0$.

De manera similar a la aproximación usada para calcular $A_i(u)$,se puede reescribir (3.16) como

$$\begin{aligned} B_{i-1}(u') &\triangleq \ln(B_{i-1}(u)) \\ &= \max_{u}(B_i(u) + \Gamma_i(u', u)) \end{aligned} \tag{3.33}$$

Finalmente, se reescribe la expresión de la LLR $a\,posteriori\ L(b_i \mid Y_1^n)$ calculada en (3.13) para el algoritmo Max-Log-MAP como

$$
\begin{aligned}
L(b_i \mid Y_1^n) &= \ln\left(\frac{\sum_{\{u',u\}\Rightarrow b_i=+1}\ \alpha_{i-1}(u')\gamma_i(u',u)\beta_i(u)}{\sum_{\{u',u\}\Rightarrow b_i=-1}\ \alpha_{i-1}(u')\gamma_i(u',u)\beta_i(u)}\right) \\
&= \max_{\{u',u\}\Rightarrow b_i=+1}(A_{i-1}(u') + \Gamma_i(u',u) + B_i(u)) \\
&\quad - \max_{\{u',u\}\Rightarrow b_i=-1}(A_{i-1}(u') + \Gamma_i(u',u) + B_i(u)) \\
&= \max_{\{u',u\}\Rightarrow b_i=+1}\boldsymbol{\sigma_i}(u',u) - \max_{\{u',u\}\Rightarrow b_i=-1}\boldsymbol{\sigma_i}(u',u) \qquad (3.34)
\end{aligned}
$$

El cálculo de la LLR se hace considerando cada transición del trellis desde el estado u' hasta el estado u, agrupando estas transiciones en las que ocurren cuando $b_i = +1$, y en las que ocurren cuando $b_i = -1$. Para ambos grupos de transiciones, se busca aquella que da el máximo valor de $A_{i-1}(u') + \Gamma_i(u',u) + B_i(u)$. Por tanto, la LLR $a\,posteriori\ L(b_i \mid Y_1^n)$ se calcula basándose únicamente en estas dos "mejores" transiciones.

3.6.2 EL ALGORITMO LOG-MAP

El algoritmo Log-MAP fue propuesto para corregir la aproximación (3.28) usada en el algoritmo Max-Log-MAP. Para que dicha aproximación sea exacta debe considerarse el logaritmo Jacobiano

$$
\begin{aligned}
\ln(e^x + e^y) &= \max(x,y) + \ln(1 + e^{-|y-x|}) \\
&= \max(x,y) + f_c(|y-x|) \qquad (3.35)
\end{aligned}
$$

donde $f_c(|y-x|)$ es el término de corrección. De manera similar a la del algoritmo Max-Log-MAP, los valores para $A_i(u) \triangleq \ln(\alpha_i(u))$ y $B_i(u) \triangleq \ln(\beta_i(u))$ se calculan con la recursión $forward$ y $backward$. Sin embargo, la maximización usada en (3.32) y (3.33) se complementa con el término de correción de (3.35). El término de correción $f_c(\delta)$ no tiene que ser calculado para cada valor de δ. Robertson $et\ al.$ encontró que sólo es necesaria una tabla que contenga ocho valores para δ, con rango entre 0 y 5.

Esto significa que el algoritmo Log-MAP es sólo un poco más complejo que el algoritmo Max-Log-MAP, pero tiene exactamente el mismo rendimiento que el algoritmo MAP.

—4—

EL ALGORITMO MAX-LOG LIST (MLLA)

EN ESTE CAPÍTULO SE ESTUDIA EL FUNCIONAMIENTO DEL ALGORITMO MAX-LOG LIST (MLLA). DICHO ALGORITMO OBTIENE PARA CÓDIGOS CONVOLUCIONALES UNA LISTA DE LAS POSIBLES SECUENCIAS DE INFORMACIÓN ENVIADA (LS) ADEMÁS DE LA DECODIFICACIÓN $SOFT$ DEL SÍMBOLO DE SALIDA. AL FINAL DEL CAPÍTULO SE MUESTRA EL FUNCIONAMIENTO DEL ALGORITMO MLLA MEDIANTE LA REALIZACIÓN DE UN EJEMPLO PRÁCTICO.

4.1 INTRODUCCIÓN

SE DEMUESTRA que el algoritmo MLLA, publicado en 2005 por Carl Fredrik Leanderson y Carl-Erik W. Sundberg (10), puede ser utilizado para obtener la combinación de la secuencia y el símbolo $soft$ de salida con una complejidad menor que algoritmos publicados previamente.

Hay dos tipos de MLLA: óptimo y subóptimo. Este último es muy adecuado para utilizar en un turbo decodificador ya que es obtenido con modificaciones del ya conocido algoritmo Max-Log MAP, algoritmo usado frecuentemente en turbo codificación. Este documento se centra en el MLLA subóptimo, pues además de ser el adecuado para turbo códigos, tiene menos complejidad que el óptimo y la LS de decodificación es cercana a la óptima.

El MLLA produce un símbolo $soft$ casi óptimo de salida igual al del algoritmo Max-Log MAP. Simultáneamente, el algoritmo produce una lista ordenada que contiene la LS-MAP estimada. Cabe señalar que únicamente se produce una secuencia con valores de símbolo $soft$, mientras que varias estimaciones de secuencias son entregadas por el algoritmo.

4.2 EL ALGORITMO MAX-LOG LIST (MLLA)

EL MLLA existe en dos versiones: el MLLA óptimo y el MLLA subóptimo.
El MLLA óptimo produce exactamente la misma lista de decodificación de la secuencia que la lista de serie del algoritmo de Viterbi (SLVA). Sin embargo, es necesario especificar el número de elementos de la lista $\mathcal{L}$ con anterioridad.
El MLLA subóptimo reduce complejidad pero sólo garantiza decodificación óptima LS-MAP para listas con tres o menos elementos. Sin embargo, para moderadas $\mathcal{L}$, la salida es cercana a la óptima.

Las operaciones del MLLA óptimo/subóptimo están muy relacionadas con el algoritmo Max-Log-MAP. La estructura del óptimo/subóptimo MLLA es tal que los valores de las sigmas $\boldsymbol{\sigma_i}(u', u)$ calculadas por el algoritmo Max-Log-MAP son reutilizadas en los cálculos de la decodificación LS.
En este documento el planteamiento será hecho, por simplicidad, únicamente para códigos convolucionales binarios, con trellis donde sólo hay dos posibles caminos parciales entrando en cada estado, tanto para el sentido $forward$ como para el sentido $backward$.

4.2.1 OPERACIONES DEL MLLA

En cada paso de la recursión $backward$ que tiene lugar en el algoritmo Max-Log-MAP, la transición más probable correspondiente a $b_i = 1$ y $b_i = 0$ es identificada y el símbolo $soft$ obtenido a la salida es la diferencia entre los correspondientes valores de $\boldsymbol{\sigma_i}(u', u)$ tal y como se expresa en (3.34). Esto implica el cálculo de $\boldsymbol{\sigma_i}(u', u)$ para todos los estados u y u' en cada paso i.

En la expresión (3.34) se observa que los valores de $\boldsymbol{\sigma_i}(u', u)$ son la suma de tres componentes:

1. la métrica $A_{i-1}(u')$ del mejor camino parcial hacia el estado u' en la dirección $forward$.

2. la métrica $B_i(u)$ del mejor camino parcial hacia el estado u en la dirección $backward$.

3. la métrica $\Gamma_i(u', u)$ de la transición entre u' y u.

Por tanto, $\boldsymbol{\sigma_i}(u', u)$ es la métrica del mejor camino global en el trellis que atraviesa la transición (u', u). Intuitivamente, parece posible identificar los caminos trellis completos con las mejores métricas de camino si se almacenan los valores de $\boldsymbol{\sigma_i}(u', u)$ calculados en cada paso i del trellis durante la recursión $backward$ en una lista ordenada.

Una observación importante es que el algoritmo Max-Log-MAP únicamente guarda las métricas $B_i(u)$ del mejor camino $backward$ parcial que entra al estado u (1.33). Por tanto, la métrica del camino de un camino trellis completo que entra al estado u' como segundo mejor camino $forward$ parcial $(A_{i-1}(u'))$ y que entra al estado u como segundo mejor camino $backward$ parcial $(B_i(u))$ no será contemplado en el conjunto de posibles métricas de caminos $\boldsymbol{\sigma_i}(u', u)$ proporcionados por el algoritmo Max-Log-MAP. Estos caminos son denominados $caminos\ de\ baja\ prioridad$.

La diferencia entre el MLLA óptimo y el subóptimo es que el MLLA subóptimo ignora los $caminos\ de\ baja\ prioridad$. Así que el MLLA subóptimo proporciona una LS de decodificación subóptima si hay un $camino\ de\ baja\ prioridad$ entre los $\mathcal{L}$ caminos más probables durante la decodificación.

La única diferencia entre la recursión $backward$ del algoritmo Max-Log-MAP y el MLLA subóptimo es el mantenimiento de una lista ordenada y actualizada en cada paso. Además de las operaciones realizadas en el algoritmo Max-Log-MAP, el MLLA subóptimo ordena las métricas $\boldsymbol{\sigma_i}(u', u)$ del segundo mejor camino que entra en cualquier estado u en el paso i.

La figura 4.1 muestra el diagrama de flujo de la recursividad $backward$ del algoritmo MLLA subóptimo. En él se muestra la manera de almacenar los valores de $\boldsymbol{\sigma_i}(u', u)$ retornados por el algoritmo Max-Log-MAP.

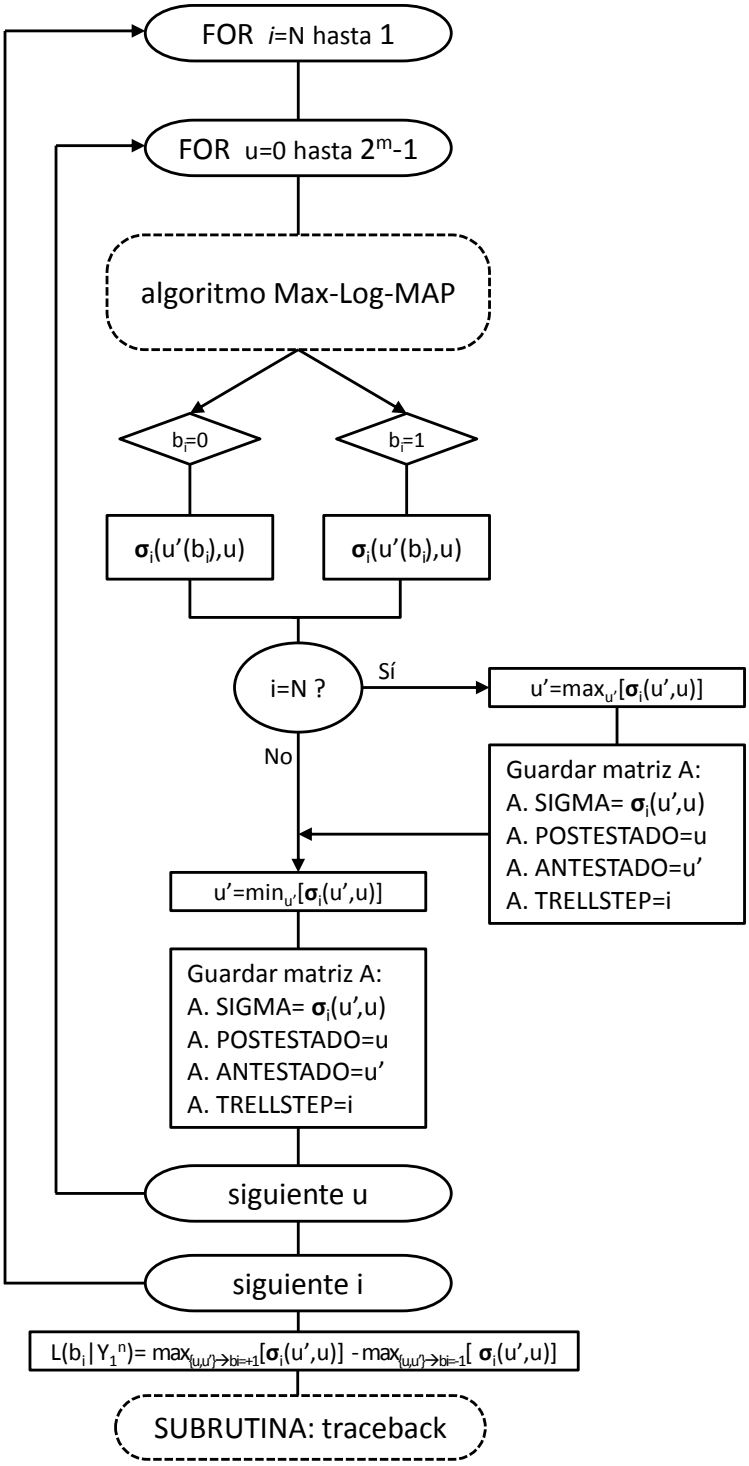

FIGURA 4.1: Diagrama de flujo de la recursividad *backward* del algoritmo MLLA subóptimo.

En cada uno de los pasos i se tienen $2^m - 1$ estados u posibles y se deben guardar los valores de $\boldsymbol{\sigma_i}(u', u)$ para todos ellos. Como se trata de un caso binario, tan sólo puede haber dos estados anteriores u' para cada estado u, cuando se ha recibido en la entrada un bit de valor 0 y cuando se ha recibido un bit de valor 1.

La métrica de transición correspondiente al mejor camino global se inserta en la lista en el primer paso i=N de la recursión $backward$. Por tanto, en cada paso $i < N$ de la recursión $backward$ es suficiente con calcular y almacenar las métricas de transición correspondientes al segundo mejor camino $forward$ parcial con el resto de las $\mathcal{L} - 1$ transiciones de la lista. Esto es debido a que la $\boldsymbol{\sigma_i}(u', u)$ del mejor camino ya ha sido calculada en el anterior paso $i + 1$ en la dirección $backward$. El almacenamiento de las $\boldsymbol{\sigma_i}(u', u)$ se hace dentro de una matriz A cuyas columnas contienen la información del valor de $\boldsymbol{\sigma_i}(u', u)$, del anterior estado, del posterior estado y del paso del trellis. La estructura de las $\mathcal{L}$ columnas de la matriz A es la siguiente:

$$A[l] = \begin{bmatrix} SIGMA \\ POSTESTADO \\ ANTESTADO \\ TRELLSTEP \end{bmatrix} = \begin{bmatrix} \boldsymbol{\sigma_i}(u', u) \\ u \\ u' \\ i \end{bmatrix}$$

Algunos registros de $\boldsymbol{\sigma_i}(u', u)$ almacenados en la matriz A pueden tener un valor igual a $-\infty$. Esto significa que representan transiciones con una probabilidad nula y, por tanto, se eliminan de la matriz.

Una vez obtenida la matriz A se reordenan sus columnas en una nueva matriz llamada R cuyas columnas cumplen la siguiente condición para todo $l = 1, 2, 3, ..., \mathcal{L}-1$:

$$R[l].SIGMA \geq R[l+1].SIGMA$$

Después de la recursión $forward$ y $backward$ del MLLA subóptimo, se tiene una matriz que contiene los registros de las transiciones correspondientes a las $\mathcal{L}$ mejores secuencias de información identificadas por el algoritmo.

A continuación se determina el símbolo $soft$ de salida para cada paso del trellis como en el algoritmo Max-Log-MAP:

$$L(b_i \mid Y_1^n) = \max_{\{u',u\} \Rightarrow b_i = +1} \boldsymbol{\sigma_i}(u', u) - \max_{\{u',u\} \Rightarrow b_i = -1} \boldsymbol{\sigma_i}(u', u) \tag{4.1}$$

De manera que se obtiene una estimación de cada bit de la palabra enviada a través del canal.

Por último se ejecuta la subrutina $traceback$. El anterior y posterior estado de cada registro son usados para recuperar la correspondiente secuencia de información. Las operaciones empiezan en el paso de trellis donde el registro de la transición ha sido originalmente insertado en la lista.

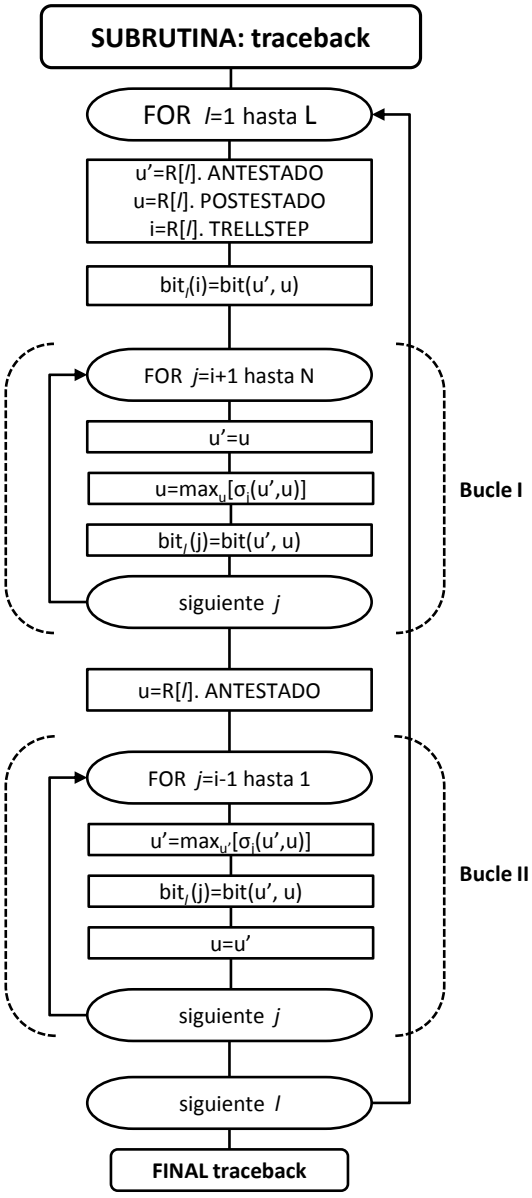

FIGURA 4.2: Diagrama de flujo de la subrutina 'TRACEBACK' usada para recuperar las secuencias de información decodificada en el algoritmo MLLA.

Tal y como se muestra en la figura 4.2, las direcciones de los caminos locales de máxima verosimilitud pueden ser recuperadas de los valores $\boldsymbol{\sigma_i}(u', u)$ ya que estos han sido guardados para cada estado y paso durante la recursión $backward$.

Para obtener la primera secuencia de información que devuelve el algoritmo MLLA se debe recuperar el registro de mayor valor guardado en la matriz R. Es decir, la transición más probable (que es la transición producida con el valor más grande de $\boldsymbol{\sigma_i}(u', u)$) que estará guardada en la posición $l = 1$ ($R[l = 1].SIGMA$). De este registro, además del valor de la sigma asociada, también se dispone del anterior estado (u'), del posterior estado (u) y del paso de trellis en el que se produce la transición (i). En un trellis, sabiendo el estado anterior y el posterior, queda determinado el bit que ha producido dicha transición, tal y como muestra la figura 4.3.

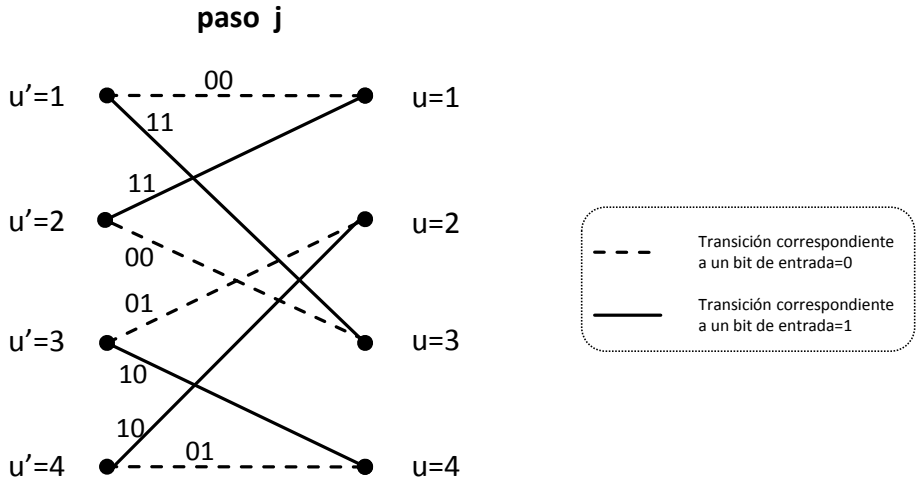

FIGURA 4.3: Posibles transiciones de un trellis de 4 estados en cualquier paso j en respuesta a los bits de entrada 0 o 1.

Por tanto, el algoritmo retorna un determinado valor (0 o 1) para la transición guardada en el primer registro de R ($bit_{l=1}(i) = 0$ o $bit_{l=1}(i) = 1$).

Después debe ejecutarse el Bucle I para todos los pasos del trellis mayores al almacenado en la matriz R ($j = i + 1, ..., N$). En dicho bucle el estado futuro almacenado en el paso anterior del trellis (u) pasa a ser el estado pasado del presente paso del trellis (u'). Para elegir el estado futuro de la presente transición se elige aquel cuya probabilidad (es decir, cuyo valor de $\boldsymbol{\sigma_i}(u', u)$) sea mayor. Una vez elegido el estado futuro adecuado (u), de nuevo tenemos definido el trellis que ha producido la transición del estado u' al estado u ($bit_{l=1}(j) = 0$ o $bit_{l=1}(j) = 1$).

Tras el desarrollo del Bucle I, se ejecuta el Bucle II, esta vez para todos los pasos del trellis menores al almacenado en la matriz R ($j = i - 1, ..., 1$). Justo antes de iniciar el Bucle II, se recupera el estado anterior (u') del paso i, ya que éste será justamente el estado futuro (u) del paso $i - 1$. Para elegir el estado pasado de la presente transición se elige aquel cuya probabilidad (es decir, cuyo valor de $\boldsymbol{\sigma_i}(u', u)$) sea mayor. Una vez elegido el estado pasado adecuado (u'), de nuevo tenemos definido el trellis que ha producido la transición u' al u($bit_{l=1}(j) = 0$ o $bit_{l=1}(j) = 1$). Para seguir ejecutando el Bucle II, hay que ir determinando el valor del futuro estado (u) del siguiente paso trellis, que se corresponderá con el anterior estado (u') del trellis actual. Por ejemplo, el siguiente paso de trellis para el que se debería buscar el valor del bit de transición sería $i - 2$. El futuro estado (u) del paso $i - 2$ será el estado anterior (u') del paso de trellis $i - 1$.

Estas operaciones se realizan de manera sucesiva hasta llegar a j=1.

Hasta aquí el algoritmo devuelve la primera secuencia $hard$ estimada por el algoritmo MLLA de longitud N.

Para obtener la segunda secuencia debe repetirse lo anteriormente explicado utilizando el segundo registro guardado en la matriz R (en la posición $l = 2$), y así sucesivamente hasta un máximo de $\mathcal{L}$ posibles secuencias estimadas.

Por tanto, ejecutando el algoritmo MLLA, se acaba obteniendo una lista de $\mathcal{L}$ posibles secuencias de información recibidas y una secuencia con valores de símbolo $soft$ (que sería la misma que devolvería el algoritmo Max-Log-MAP).

4.3 EJEMPLO DE DECODIFICACIÓN MLLA

EL ALGORITMO MLLA se aplica para recuperar la palabra enviada ($x_C = [110111]$) mediante la decodificación MLLA de la palabra recibida tras atravesar el canal (y_R). El esquema de codificación y decodificación de la palabra x_C es el mostrado a continuación:

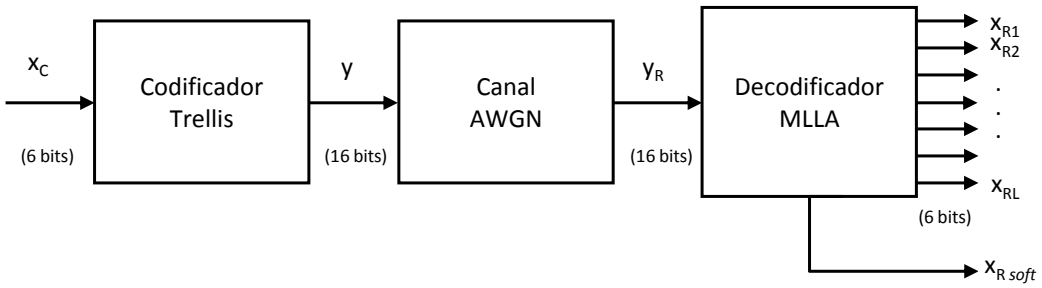

FIGURA 4.4: Esquema del ejemplo de decodificación MLLA.

Cada uno de los bloques se explica en los siguientes apartados.

4.3.1 CODIFICADOR TRELLIS

Este primer bloque es un codificador convolucional sistemático y recursivo (RSC) de razón r=$\frac{1}{2}$ cuyas transiciones de estados y bits de salida correspondientes se muestran a continuación.

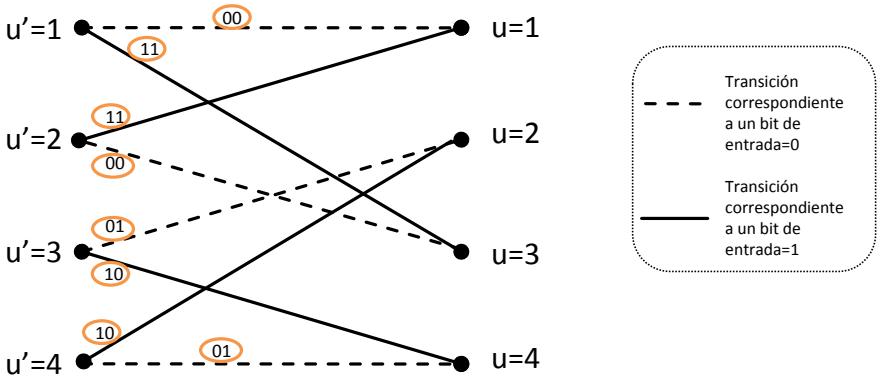

FIGURA 4.5: Transiciones y bits de salida de un trellis de 4 estados.

La palabra enviada, $x_C = [110111]$, se codifica mediante el trellis dibujado en la figura 4.6.

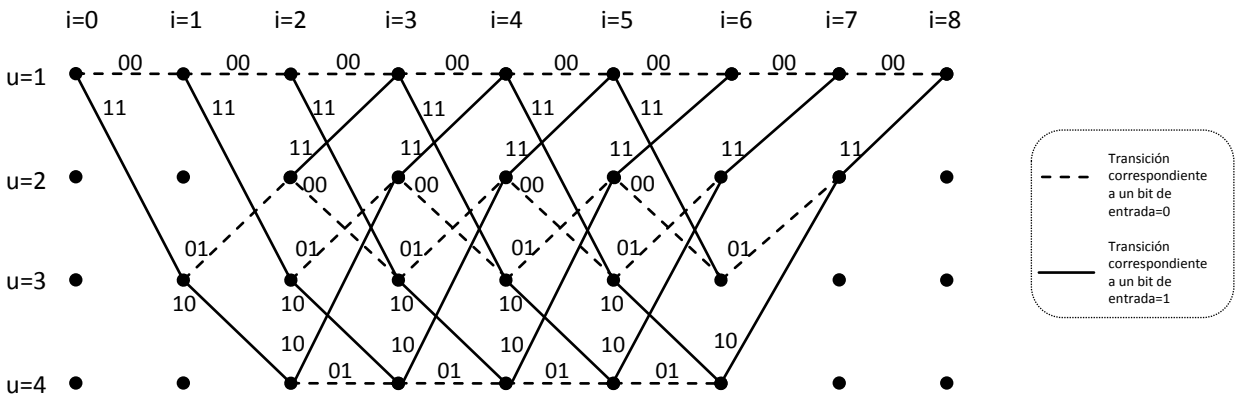

FIGURA 4.6: Diagrama de trellis de 4 estados.

Para poder decodificar una palabra con el algoritmo MLLA, la palabra codificada debe empezar y acabar en el estado inicial. En este caso, la palabra deberá empezar y acabar en el estado $u = 1$. Para ello se añaden dos bits de cola que serán, según indica el trellis, los bits '01'.

Por tanto, como muestra la figura 4.7, la palabra a codificar es finalmente $x = [11011101]$ y la palabra obtenida a la salida del codificador convolucional será $y = [1110011011110111]$.

En un último paso, el codificador envía la palabra resultante por el canal usando el alfabeto polar en lugar del binario. Así que la palabra a la salida del codificador es $y = [1\,1\,1 - 1 - 1\,1\,1 - 1\,1\,1\,1\,1 - 1\,1\,1\,1]$.

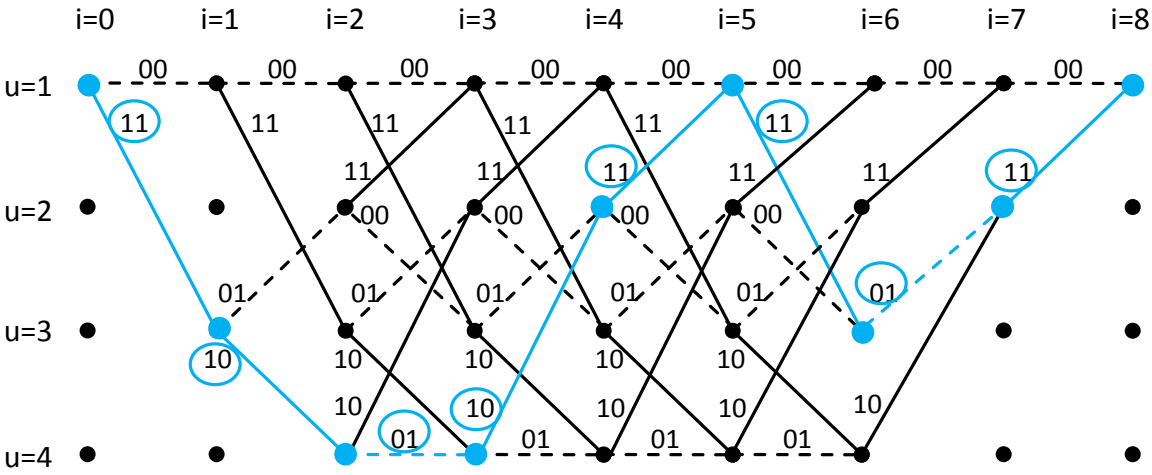

FIGURA 4.7: Codificación de la palabra 110111.

4.3.2 CANAL AWGN

El ejemplo se realiza utilizando un canal AWGN (Additive White Gaussian Noise) que presenta un ruido blanco, aditivo y gaussiano.

La relación entre la potencia media de la señal de información y la potencia media de la señal de ruido es $(S/N) = 10\,dB$.

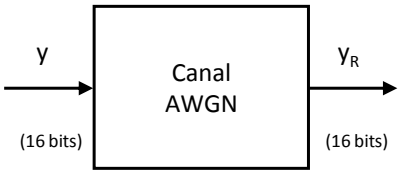

FIGURA 4.8: Canal AWGN.

La palabra $y = [1\,1\,1 - 1 - 1\,1\,1 - 1\,1\,1\,1\,1 - 1\,1\,1\,1]$ es introducida en el canal y a la salida se recibe la palabra $y_R = [\ 0,838 \quad 1,079 \quad 1,117 \quad$ -0,943 $\quad$ -1,012 $\quad 0,493 \quad 1,107 \quad$ -1,041 $\quad 1,153 \quad 1,189 \quad 0,973 \quad 1,103 \quad$ -1,106 $\quad 0,898 \quad 0,879 \quad 0,699]$.

4.3.3 DECODIFICADOR MLLA

Para realizar la decodificación MLLA se deben seguir los pasos indicados en el diagrama de la figura 4.1 visto en el capítulo 4.2.1.

Tal y como se indica en el diagrama, lo primero que debe calcularse son los valores de $\boldsymbol{\sigma_i}(u', u)$ retornados por el **algoritmo Max-Log-MAP** para los 4 estados y los 8 pasos del trellis.

Según la expresión (3.34) el valor de $\boldsymbol{\sigma_i}(u', u)$ cuando se recibe un bit de valor $b_i = 1$ o $b_i = -1$ es:

$$\boldsymbol{\sigma}_i(u', u) = A_{i-1}(u') + \Gamma_i(u', u) + B_i(u) \tag{4.2}$$

Para encontrar estos valores se calculan primero todos los valores de $\Gamma_i(u', u)$, después todos los de $A_{i-1}(u')$ y finalmente todos los valores de $B_i(u)$.

1. CÁCULO DE LOS VALORES DE $\Gamma_i(u', u)$.

   Los valores de $\gamma_i(u', u)$ se calculan como muestra la expresión (3.24) que pertenece al capítulo 3.4.5.

$$\gamma_i(u', u) = Ce^{(b_i L(b_i))/2} e^{\frac{L_c}{2} y_{i1} x_{i1}} e^{\frac{L_c}{2} \sum_{k=2}^{n} (y_{ik} x_{ik})} \tag{4.3}$$

   El valor de la constante de canal se fija en este ejemplo con valor $L_c = 2$ y la información a priori (como no hay dos decodificadores -como los habría en el caso del turbo decodificador- no hay iteraciones con transmisión de información) es $L(b_i) = 0$.

   Por otro lado, como el codificador es de razón r=$\frac{1}{2}$, cada bit de entrada tendrá a la salida un bit sistemático más un bit de paridad ($n = 2$).

   Los valores de x_{ik}, donde $k = (1, 2)$, son los valores teóricos que se deberían recibir a la salida en cada paso i del trellis, $i = (1, ..., 8)$. Estos valores se calculan, fijando el estado pasado u' y el bit que provoca la transición (1 o -1), mediante el diagrama de transición del trellis de la figura(4.5). El valor x_{i1} corresponde al bit sistemático, x_{is}, y x_{i2} corresponde al bit de paridad, x_{ip}.

   Los valores de y_{ik}, , donde $k = (1, 2)$, corresponden a la palabra recibida a través del canal en cada paso i del trellis, $i = (1, ..., 8)$. El valor y_{i1} corresponde al bit sistemático, y_{is}, y y_{i2} corresponde al bit de paridad, y_{ip}.

   En este ejemplo se ha recibido:

   $y_R = [\ 0,838 \quad 1,079 \quad 1,117 \quad$ -0,943 $\quad$ -1,012 $\quad 0,493 \quad 1,107 \quad$ -1,041
   $1,153 \quad 1,189 \quad 0,973 \quad 1,103 \quad$ -1,106 $\quad 0,898 \quad 0,879 \quad 0,699]$

   así que los valores definidos con anterioridad serán:

$y_{1s} = 0,838$	$y_{2s} = 1,117$	y_{3s} =-1,012	$y_{4s} = 1,107$	$y_{5s} = 1,153$	$y_{6s} = 0,973$	y_{7s} =-1,106	$y_{8s} = 0,879$
$y_{1p} = 1,079$	y_{2p} =-0,943	$y_{3p} = 0,493$	y_{4p} =-1,041	$y_{5p} = 1,189$	$y_{6p} = 1,103$	$y_{7p} = 0,898$	$y_{8p} = 0,699$

   Con estos datos se puede reescribir $\gamma_i(u', u)$ de la siguiente manera:

$$\begin{aligned} \gamma_i(u', u) &= Ce^0 e^{\frac{2}{2} y_{i1} x_{i1}} e^{\frac{2}{2} (y_{i2} x_{i2})} \\ &= Ce^{y_{is} x_{is}} e^{y_{ip} x_{ip}} \\ &= Ce^{y_{is} x_{is} + y_{ip} x_{ip}} \end{aligned} \tag{4.4}$$

Finalmente se puede hallar el valor de $\Gamma_i(u', u)$ mediante la relación hallada en la expresión (3.31)

$$\Gamma_i(u', u) \triangleq \ln(\gamma_i(u', u)) = ln(Ce^{y_{is}x_{is}+y_{ip}x_{ip}}) = K(y_{is}x_{is} + y_{ip}x_{ip}) \tag{4.5}$$

donde K=ln(C) es una constante cuyo valor no tendrá importancia al realizar las comparaciones, así que será obviado para la realización de los cálculos.

(a) VALORES DE $\Gamma_i(u', u)$ CUANDO SE HA RECIBIDO $b_i = -1$.

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u', u) = y_{is}x_{is} + y_{ip}x_{ip}$
1	0,838	1,079	1	1	-1	-1	-1,917
			2	3	-1	-1	-1,917
			3	2	-1	1	0,241
			4	4	-1	1	0,241

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u', u) = y_{is}x_{is} + y_{ip}x_{ip}$
2	1,117	-0,943	1	1	-1	-1	-0,174
			2	3	-1	-1	-0,174
			3	2	-1	1	-2,060
			4	4	-1	1	-2,060

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u', u) = y_{is}x_{is} + y_{ip}x_{ip}$
3	-1,012	0,493	1	1	-1	-1	0,519
			2	3	-1	-1	0,519
			3	2	-1	1	1,505
			4	4	-1	1	1,505

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u', u) = y_{is}x_{is} + y_{ip}x_{ip}$
4	1,107	-1,041	1	1	-1	-1	-0,066
			2	3	-1	-1	-0,066
			3	2	-1	1	-2,148
			4	4	-1	1	-2,148

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u', u) = y_{is}x_{is} + y_{ip}x_{ip}$
5	1,153	1,189	1	1	-1	-1	-2,342
			2	3	-1	-1	-2,342
			3	2	-1	1	0,036
			4	4	-1	1	0,036

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
6	0,973	1,103	1	1	-1	-1	-2,076
			2	3	-1	-1	-2,076
			3	2	-1	1	0,130
			4	4	-1	1	0,130

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
7	-1,106	0,898	1	1	-1	-1	0,208
			2	3	-1	-1	0,208
			3	2	-1	1	2,004
			4	4	-1	1	2,004

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
8	0,879	0,699	1	1	-1	-1	-1,578
			2	3	-1	-1	-1,578
			3	2	-1	1	-0,180
			4	4	-1	1	-0,180

TABLA 4.1: Tablas de valores de Γ_{i_0} para cada paso del trellis

(b) VALORES DE $\Gamma_i(u', u)$ CUANDO SE HA RECIBIDO $b_i = +1$.

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
1	0,838	1,079	1	3	1	1	1,917
			2	1	1	1	1,917
			3	4	1	-1	-0,241
			4	2	1	-1	-0,241

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
2	1,117	-0,943	1	3	1	1	0,174
			2	1	1	1	0,174
			3	4	1	-1	2,060
			4	2	1	-1	2,060

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
3	-1,012	0,493	1	3	1	1	-0,519
			2	1	1	1	-0,519
			3	4	1	-1	-1,505
			4	2	1	-1	-1,505

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
4	1,107	-1,041	1	3	1	1	0,066
			2	1	1	1	0,066
			3	4	1	-1	2,148
			4	2	1	-1	2,148

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
5	1,153	1,189	1	3	1	1	2,342
			2	1	1	1	2,342
			3	4	1	-1	-0,036
			4	2	1	-1	-0,036

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
6	0,973	1,103	1	3	1	1	2,076
			2	1	1	1	2,076
			3	4	1	-1	-0,130
			4	2	1	-1	-0,130

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
7	-1,106	0,898	1	3	1	1	-0,208
			2	1	1	1	-0,208
			3	4	1	-1	-2,004
			4	2	1	-1	-2,004

i	y_{is}	y_{ip}	u'	u	x_{is}	x_{ip}	$\Gamma_{i_0}(u',u) = y_{is}x_{is} + y_{ip}x_{ip}$
8	0,879	0,699	1	3	1	1	1,578
			2	1	1	1	1,578
			3	4	1	-1	0,180
			4	2	1	-1	0,180

TABLA 4.2: Tablas de valores de Γ_{i_1} para cada paso del trellis

Resumiendo, los valores de $\Gamma_{i_0}(u', u)$ y $\Gamma_{i_1}(u', u)$ obtenidos son:

$\Gamma_{i_0}(u', u)$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$	$i = 8$
$u' = 1$	-1,917	-0,174	0,519	-0,066	-2,342	-2,076	0,208	-1,578
$u' = 2$	-1,917	-0,174	0,519	-0,066	-2,342	-2,076	0,208	-1,578
$u' = 3$	0,241	-2,060	1,505	-2,148	0,036	0,130	2,004	-0,180
$u' = 4$	0,241	-2,060	1,505	-2,148	0,036	0,130	2,004	-0,180

TABLA 4.3: Tabla resumen de los valores de Γ_{i_0}

$\Gamma_{i_1}(u', u)$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$	$i = 8$
$u' = 1$	1,197	0,174	-0,519	0,066	2,342	2,076	-0,208	1,578
$u' = 2$	1,197	0,174	-0,519	0,066	2,342	2,076	-0,208	1,578
$u' = 3$	-0,241	2,060	-1,505	2,148	-0,036	-0,130	-2,004	0,180
$u' = 4$	-0,241	2,060	-1,505	2,148	-0,036	-0,130	-2,004	0,180

TABLA 4.4: Tabla resumen de los valores de Γ_{i_1}

2. CÁCULO DE LOS VALORES DE $A_{i-1}(u')$.

   Los valores de $A_i(u)$ se calculan como muestra la expresión (3.32) que pertenece al capítulo 3

$$A_i(u) = \max_{u'}(A_{i-1}(u') + \Gamma_i(u', u)) \tag{4.6}$$

   teniendo en cuenta que las condiciones iniciales son:

$$\begin{aligned} A_{i=0}(u = 1) &= 0 \\ A_{i=0}(u) &= -\infty \quad si \;\; u \neq 1 \end{aligned} \tag{4.7}$$

   Para realizar los cálculos, se fija el paso del trellis i y el estado futuro u. Entonces, según el bit que se haya recibido a la entrada, se tiene un determinado estado pasado u' y se obtienen los valores de $A_i(u)$.

		$bit(i)=0$				$bit(i)=1$				
i	u	u'	$A_{i-1}(u')$	$\Gamma_{i_0}(u',u)$	$A_{i-1}+\Gamma_{i_0}$	u'	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$A_{i-1}+\Gamma_{i_1}$	$A_i(u)$
1	1	1	0	-1,917	-1,917	2	-∞	1,917	-∞	-1,917
	2	3	-∞	0,241	-∞	4	-∞	-0,241	-∞	-∞
	3	2	-∞	-1,917	-∞	1	0	1,917	1,917	1,917
	4	4	-∞	0,241	-∞	3	-∞	-0,241	-∞	-∞

		$bit(i)=0$				$bit(i)=1$				
i	u	u'	$A_{i-1}(u')$	$\Gamma_{i_0}(u',u)$	$A_{i-1}+\Gamma_{i_0}$	u'	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$A_{i-1}+\Gamma_{i_1}$	$A_i(u)$
2	1	1	-1,917	-0,174	-2,091	2	-∞	0,174	-∞	-2,091
	2	3	1,917	-2,060	-0,143	4	-∞	2,060	-∞	-0,143
	3	2	-∞	-0,174	-∞	1	-1,917	0,174	-1,743	-1,743
	4	4	-∞	-2,060	-∞	3	1,917	2,060	3,977	3,977

		$bit(i)=0$				$bit(i)=1$				
i	u	u'	$A_{i-1}(u')$	$\Gamma_{i_0}(u',u)$	$A_{i-1}+\Gamma_{i_0}$	u'	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$A_{i-1}+\Gamma_{i_1}$	$A_i(u)$
3	1	1	-2,091	0,519	-1,572	2	-0,143	-0,519	-0,662	-0,662
	2	3	-1,743	1,505	-0,238	4	3,977	-1,505	2,472	2,472
	3	2	-0,143	0,519	0,376	1	-2,091	-0,519	-2,610	0,376
	4	4	3,977	1,505	5,482	3	-1,743	-1,505	-3,248	5,482

		$bit(i)=0$				$bit(i)=1$				
i	u	u'	$A_{i-1}(u')$	$\Gamma_{i_0}(u',u)$	$A_{i-1}+\Gamma_{i_0}$	u'	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$A_{i-1}+\Gamma_{i_1}$	$A_i(u)$
4	1	1	-0,662	-0,066	-0,728	2	2,472	0,066	2,538	2,538
	2	3	0,376	-2,148	-1,772	4	5,482	2,148	7,630	7,630
	3	2	2,472	-0,066	2,406	1	-0,662	0,066	-0,596	2,406
	4	4	5,482	-2,148	3,334	3	0,376	2,148	2,524	3,334

		$bit(i)=0$				$bit(i)=1$				
i	u	u'	$A_{i-1}(u')$	$\Gamma_{i_0}(u',u)$	$A_{i-1}+\Gamma_{i_0}$	u'	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$A_{i-1}+\Gamma_{i_1}$	$A_i(u)$
5	1	1	2,538	-2,342	0,196	2	7,630	2,342	9,972	9,972
	2	3	2,406	0,036	2,442	4	3,334	-0,036	3,298	3,298
	3	2	7,630	-2,342	5,288	1	2,538	2,342	4,880	5,288
	4	4	3,334	0,036	3,370	3	2,406	-0,036	2,370	3,370

		$bit(i)=0$				$bit(i)=1$				
i	u	u'	$A_{i-1}(u')$	$\Gamma_{i_0}(u',u)$	$A_{i-1}+\Gamma_{i_0}$	u'	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$A_{i-1}+\Gamma_{i_1}$	$A_i(u)$
6	1	1	9,972	-2,076	7,896	2	3,298	2,076	5,374	7,896
	2	3	5,288	0,130	5,418	4	3,370	-0,130	3,240	5,418
	3	2	3,298	-2,076	1,222	1	9,972	2,076	12,048	12,048
	4	4	3,370	0,130	3,500	3	5,288	-0,130	5,158	5,158

		$bit(i)=0$				$bit(i)=1$				
i	u	u'	$A_{i-1}(u')$	$\Gamma_{i_0}(u',u)$	$A_{i-1}+\Gamma_{i_0}$	u'	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$A_{i-1}+\Gamma_{i_1}$	$A_i(u)$
7	1	1	7,896	0,208	8,104	2	5,418	-0,208	5,210	8,104
	2	3	12,048	2,004	14,052	4	5,158	-2,004	3,154	14,052
	3	2	5,418	0,208	5,626	1	7,896	-0,208	7,688	7,688
	4	4	5,158	2,004	7,162	3	12,048	-2,004	10,044	10,044

		$bit(i)=0$				$bit(i)=1$				
i	u	u'	$A_{i-1}(u')$	$\Gamma_{i_0}(u',u)$	$A_{i-1}+\Gamma_{i_0}$	u'	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$A_{i-1}+\Gamma_{i_1}$	$A_i(u)$
8	1	1	8,104	-1,578	6,526	2	14,052	1,578	15,630	15,630
	2	3	7,688	-0,180	7,508	4	10,044	0,180	10,224	10,224
	3	2	14,052	-1,578	12,474	1	8,104	1,578	9,682	12,474
	4	4	10,044	-0,180	9,864	3	7,688	0,180	7,868	9,864

TABLA 4.5: Tablas de valores de A_i para cada paso del trellis

Resumiendo, los valores de $A_i(u)$ obtenidos son:

$A_i(u)$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$
$u=1$	0	-1,917	-2,091	-0,662	2,538	9,972	7,896	8,104	15,630
$u=2$	-∞	-∞	-0,143	2,472	7,630	3,298	5,418	14,052	10,224
$u=3$	-∞	1,917	-1,743	0,376	2,406	5,288	12,048	7,688	12,474
$u=4$	-∞	-∞	3,977	5,482	3,334	3,370	5,158	10,044	9,864

TABLA 4.6: Tabla resumen de los valores de A_i

3. CÁCULO DE LOS VALORES DE $B_i(u)$.

Los valores de $B_i(u)$ se calculan como muestra la expresión (3.33) que pertenece al capítulo 3

$$\begin{aligned} B_{i-1}(u') &= \max_u(B_i(u) + \Gamma_i(u', u)) \\ B_i(u') &= \max_u(B_{i+1}(u) + \Gamma_{i+1}(u', u)) \end{aligned} \tag{4.8}$$

teniendo en cuenta que las condiciones iniciales son:

$$\begin{aligned} B_{i=8}(u = 1) &= 0 \\ B_{i=8}(u) &= -\infty \quad si \;\; u \neq 1 \end{aligned} \tag{4.9}$$

Para realizar los cálculos, se fija el paso del trellis i y el estado pasado u'. Entonces, según el bit que se haya recibido a la entrada, se tiene un determinado estado futuro u y se obtienen los valores de $B_i(u)$.

		$bit(i) = 0$				$bit(i) = 1$				
i	u'	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_0}(u', u)$	$B + \Gamma$	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_1}(u', u)$	$B + \Gamma$	$B_i(u')$
7	1	1	0	-1,578	-1,578	3	-∞	1,578	-∞	-1,578
	2	3	-∞	-1,578	-∞	1	0	1,578	1,578	1,578
	3	2	-∞	-0,180	-∞	4	-∞	0,180	-∞	-∞
	4	4	-∞	-0,180	-∞	2	-∞	0,180	-∞	-∞

		$bit(i) = 0$				$bit(i) = 1$				
i	u'	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_0}(u', u)$	$B + \Gamma$	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_1}(u', u)$	$B + \Gamma$	$B_i(u')$
6	1	1	-1,578	0,208	-1,370	3	-∞	- 0,208	-∞	-1,370
	2	3	-∞	0,208	-∞	1	-1,578	-0,208	-1,786	-1,786
	3	2	1,578	2,004	3,582	4	-∞	-2,004	-∞	3,582
	4	4	-∞	2,004	-∞	2	1,578	-2,004	-0,426	-0,426

		$bit(i) = 0$				$bit(i) = 1$				
i	u'	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_0}(u', u)$	$B + \Gamma$	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_1}(u', u)$	$B + \Gamma$	$B_i(u')$
5	1	1	-1,370	-2,076	-3,446	3	3,582	2,076	5,658	5,658
	2	3	3,582	-2,076	1,506	1	-1,370	2,076	0,706	1,506
	3	2	-1,786	0,130	-1,656	4	-0,426	-0,130	-0,556	-0,556
	4	4	-0,426	0,130	-0,296	2	-1,786	-0,130	-1,916	-0,296

		$bit(i)=0$				$bit(i)=1$				
i	u'	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_0}(u',u)$	$B+\Gamma$	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_1}(u',u)$	$B+\Gamma$	$B_i(u')$
4	1	1	5,658	-2,342	3,316	3	-0,556	2,342	1,786	3,316
	2	3	-0,556	-2,342	-2,898	1	5,658	2,342	8,000	8,000
	3	2	1,506	0,036	1,542	4	-0,296	-0,036	-0,332	1,542
	4	4	-0,296	0,036	-0,260	2	1,506	-0,036	1,470	1,470

		$bit(i)=0$				$bit(i)=1$				
i	u'	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_0}(u',u)$	$B+\Gamma$	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_1}(u',u)$	$B+\Gamma$	$B_i(u')$
3	1	1	3,316	-0,066	3,250	3	1,542	0,066	1,608	3,250
	2	3	1,542	-0,066	1,476	1	3,316	0,066	3,382	3,382
	3	2	8,000	-2,148	5,852	4	1,470	2,148	3,618	5,852
	4	4	1,470	-2,148	-0,678	2	8,000	2,148	10,148	10,148

		$bit(i)=0$				$bit(i)=1$				
i	u'	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_0}(u',u)$	$B+\Gamma$	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_1}(u',u)$	$B+\Gamma$	$B_i(u')$
2	1	1	3,250	0,519	3,769	3	5,852	-0,519	5,333	5,333
	2	3	5,852	0,519	6,371	1	3,250	-0,519	2,731	6,371
	3	2	3,382	1,505	4,887	4	10,148	-1,505	8,643	8,643
	4	4	10,148	1,505	11,653	2	3,382	-1,505	1,877	11,653

		$bit(i)=0$				$bit(i)=1$				
i	u'	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_0}(u',u)$	$B+\Gamma$	u	$B_{i+1}(u)$	$\Gamma_{(i+1)_1}(u',u)$	$B+\Gamma$	$B_i(u')$
1	1	1	5,333	-0,174	5,159	3	8,643	0,174	8,817	8,817
	2	3	8,643	-0,174	8,469	1	5,333	0,174	5,507	8,469
	3	2	6,371	-2,060	4,311	4	11,653	2,060	13,713	13,713
	4	4	11,653	-2,060	9,593	2	6,371	2,060	8,431	9,593

TABLA 4.7: Tablas de valores de B_i para cada paso del trellis

Resumiendo, los valores de $B_i(u')$ obtenidos son:

$B_i(u')$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$
$u'=1$	8,817	5,333	3,250	3,316	5,658	-1,370	-1,578	0
$u'=2$	8,469	6,371	3,382	8,000	1,506	-1,786	1,578	-∞
$u'=3$	13,713	8,643	5,852	1,542	-0,556	3,582	-∞	-∞
$u'=4$	9,593	11,653	10,148	1,470	-0,296	-0,426	-∞	-∞

TABLA 4.8: Tabla resumen de los valores de B_i

4. CÁCULO DE LOS VALORES DE $\boldsymbol{\sigma_i}(u', u)$.

Los valores de $\boldsymbol{\sigma_i}(u', u)$ se calculan como muestra la expresión (3.34) que pertenece al capítulo 3. A continuación se reorganiza dicha fórmula con la nomenclatura usada en este ejemplo.

$$\begin{aligned} L(b_i \mid Y_1^n) &= \max_{\{u',u\}\Rightarrow b_i=+1}(A_{i-1}(u') + \Gamma_i(u',u) + B_i(u)) \\ &\quad - \max_{\{u',u\}\Rightarrow b_i=-1}(A_{i-1}(u') + \Gamma_i(u',u) + B_i(u)) \\ &= \max_{\{u',u\}\Rightarrow b_i=+1} \boldsymbol{\sigma_i}(u',u) - \max_{\{u',u\}\Rightarrow b_i=-1} \boldsymbol{\sigma_i}(u',u) \\ &= \max_{\{u',u\}\Rightarrow b_i=+1} \boldsymbol{\sigma_{i_1}}(u',u) - \max_{\{u',u\}\Rightarrow b_i=-1} \boldsymbol{\sigma_{i_0}}(u',u) \\ &= \max_{\{u',u\}\Rightarrow b_i=+1}(A_{i-1}(u') + \Gamma_{i_1}(u',u) + B_i(u)) \\ &\quad - \max_{\{u',u\}\Rightarrow b_i=-1}(A_{i-1}(u') + \Gamma_{i_0}(u',u) + B_i(u)) \end{aligned} \tag{4.10}$$

Para realizar los cálculos, se fija el paso del trellis i y el estado pasado u'. Entonces, según el bit que se haya recibido a la entrada, se tiene un determinado estado futuro u y se obtienen los valores de $\boldsymbol{\sigma_i}(u', u)$.

(a) VALORES DE $\boldsymbol{\sigma_i}(u', u)$ CUANDO SE HA RECIBIDO $b_i = -1$.
Los valores de $\boldsymbol{\sigma_i}(u', u)$ cuando se ha recibido $b_i = -1$ se calculan mediante la expresión:

$$\boldsymbol{\sigma_{i_0}}(u', u) \quad = \quad A_{i-1}(u') + \Gamma_{i_0}(u', u) + B_i(u) \tag{4.11}$$

para cada paso i del trellis y cada posible estado anterior u'.

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_0}(u', u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_0}}(u', u)$
1	1	1	0	-1,917	8,817	6,9
	2	3	-∞	-1,917	13,713	-∞
	3	2	-∞	0,241	8,469	-∞
	4	4	-∞	0,241	9,593	-∞

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_0}(u', u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_0}}(u', u)$
2	1	1	-1,917	-0,174	5,333	3,242
	2	3	-∞	-0,174	8,643	-∞
	3	2	1,917	-2,060	6,371	6,228
	4	4	-∞	-2,060	11,653	-∞

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_0}(u', u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_0}}(u', u)$
3	1	1	-2,091	0,519	3,250	1,678
	2	3	-0,143	0,519	5,852	6,228
	3	2	-1,743	1,505	3,382	3,144
	4	4	3,977	1,505	10,148	15,630

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_0}(u', u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_0}}(u', u)$
4	1	1	-0,662	-0,066	3,316	2,588
	2	3	2,472	-0,066	1,542	3,948
	3	2	0,376	-2,148	8,000	6,228
	4	4	5,482	-2,148	1,470	4,804

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_0}(u', u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_0}}(u', u)$
5	1	1	2,538	-2,342	5,658	5,854
	2	3	7,630	-2,342	-0,556	4,732
	3	2	2,406	0,036	1,506	3,948
	4	4	3,334	0,036	-0,296	3,074

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_0}(u', u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_0}}(u', u)$
6	1	1	9,972	-2,076	-1,370	6,526
	2	3	3,298	-2,076	3,582	4,804
	3	2	5,288	0,130	-1,786	3,632
	4	4	3,370	0,130	-0,426	3,074

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_0}(u',u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_0}}(u',u)$
7	1	1	7,896	0,208	-1,578	6,526
	2	3	5,418	0,208	-∞	-∞
	3	2	12,048	2,004	1,578	15,630
	4	4	5,158	2,004	-∞	-∞

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_0}(u',u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_0}}(u',u)$
8	1	1	8,104	-1,578	0	6,526
	2	3	14,052	-1,578	-∞	-∞
	3	2	7,688	-0,180	-∞	-∞
	4	4	10,044	-0,180	-∞	-∞

TABLA 4.9: Tablas de valores de $\boldsymbol{\sigma_{i_0}}(u',u)$ para cada paso del trellis

(b) VALORES DE $\boldsymbol{\sigma_i}(u',u)$ CUANDO SE HA RECIBIDO $b_i = +1$.
Los valores de $\boldsymbol{\sigma_i}(u',u)$ cuando se ha recibido $b_i = +1$ se calculan mediante la expresión:

$$\boldsymbol{\sigma_{i_1}}(u',u) \;=\; A_{i-1}(u') + \Gamma_{i_1}(u',u) + B_i(u) \tag{4.12}$$

para cada paso i del trellis y cada posible estado anterior u'.

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_1}}(u',u)$
1	1	3	0	1,917	13,713	15,630
	2	1	-∞	1,917	8,817	-∞
	3	4	-∞	-0,241	9,593	-∞
	4	2	-∞	-0,241	8,469	-∞

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_1}}(u',u)$
2	1	3	-1,917	0,174	8,643	6,900
	2	1	-∞	0,174	5,333	-∞
	3	4	1,917	2,060	11,653	15,630
	4	2	-∞	2,060	6,371	-∞

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_1}}(u',u)$
3	1	3	-2,091	-0,519	5,852	3,242
	2	1	-0,143	-0,519	3,250	2,588
	3	4	-1,743	-1,505	10,148	6,900
	4	2	3,977	-1,505	3,382	5,854

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_1}}(u',u)$
4	1	3	-0,662	0,066	1,542	0,946
	2	1	2,472	0,066	3,316	5,854
	3	4	0,376	2,148	1,470	3,994
	4	2	5,482	2,148	8,000	15,630

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_1}}(u',u)$
5	1	3	2,538	2,342	-0,556	4,324
	2	1	7,630	2,342	5,658	15,63
	3	4	2,406	-0,036	-0,296	2,074
	4	2	3,334	-0,036	1,506	4,804

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_1}}(u',u)$
6	1	3	9,972	2,076	3,582	15,630
	2	1	3,298	2,076	-1,370	4,004
	3	4	5,288	-0,130	-0,426	4,732
	4	2	3,370	-0,130	-1,786	1,454

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_1}}(u',u)$
7	1	3	7,896	-0,208	-∞	-∞
	2	1	5,418	-0,208	-1,578	3,632
	3	4	12,048	-2,004	-∞	-∞
	4	2	5,158	-2,004	1,578	4,732

i	u'	u	$A_{i-1}(u')$	$\Gamma_{i_1}(u',u)$	$B_i(u)$	$\boldsymbol{\sigma_{i_1}}(u',u)$
8	1	3	8,104	1,578	-∞	-∞
	2	1	14,052	1,578	0	15,63
	3	4	7,688	0,180	-∞	-∞
	4	2	10,044	0,180	-∞	-∞

TABLA 4.10: Tablas de valores de $\boldsymbol{\sigma_{i_1}}(u',u)$ para cada paso del trellis

Resumiendo, los valores de $\boldsymbol{\sigma_{i_0}}(u', u)$ y $\boldsymbol{\sigma_{i_1}}(u', u)$ obtenidos son:

$\boldsymbol{\sigma_{i_0}}(u', u)$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$	$i = 8$
$u' = 1$	6,900	3,242	1,678	2,588	5,854	6,526	6,526	6,526
$u' = 2$	-∞	-∞	6,228	3,948	4,732	4,804	-∞	-∞
$u' = 3$	-∞	6,228	3,144	6,228	3,948	3,632	15,630	-∞
$u' = 4$	-∞	-∞	15,630	4,804	3,074	3,074	-∞	-∞

TABLA 4.11: Tabla resumen de los valores de $\boldsymbol{\sigma_{i_0}}(u', u)$

$\boldsymbol{\sigma_{i_1}}(u', u)$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$	$i = 8$
$u' = 1$	15,630	6,900	3,242	0,946	4,324	15,630	-∞	-∞
$u' = 2$	-∞	-∞	2,588	5,854	15,630	4,004	3,632	15,630
$u' = 3$	-∞	15,630	6,900	3,994	2,074	4,732	-∞	-∞
$u' = 4$	-∞	-∞	5,854	15,630	4,804	1,454	4,732	-∞

TABLA 4.12: Tabla resumen de los valores de $\boldsymbol{\sigma_{i_1}}(u', u)$

Aquí finaliza la obtención de las métricas entregadas por el **algoritmo Max-Log-MAP**. A continuación, el MLLA guarda las métricas $\boldsymbol{\sigma_i}(u', u)$ del segundo mejor camino que entra en cualquier estado u en el paso i en una matriz llamada A.

El almacenamiento de valores en la matriz A se realiza como indica la figura 4.1. Es decir, desde los pasos del trellis $i = 8$ hasta $i = 1$, para cada estado $u = 1$, $u = 2$, $u = 3$ y $u = 4$, se deben mirar los posibles estados anteriores u' (dependiendo de si el bit que provoca la transición es $b(i) = 0$ o $b(i) = 1$) que correspondan según el trellis, y se elige el de menor valor de $\boldsymbol{\sigma_i}(u', u)$ para almacenar en la **matriz** A.

En el caso del paso i=8, se guardan tanto el valor máximo como el mínimo de $\boldsymbol{\sigma_i}(u', u)$.

A continuación se muestran las comparaciones hechas para obtener los valores de las métricas que formarán la matriz A.

				$i=8$	$i=7$	$i=6$	$i=5$
$u=1$	$b(i)=0$	$u'=1$	$\boldsymbol{\sigma_{i_0}}(u',u)$	6,526	6,526	6,526	5,854
	$b(i)=1$	$u'=2$	$\boldsymbol{\sigma_{i_1}}(u',u)$	15,630	3,632	4,004	15,630
	$A=min(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			6,526	3,632	4,004	5,854
	$A=max(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			15,630			

				$i=4$	$i=3$	$i=2$	$i=1$
$u=1$	$b(i)=0$	$u'=1$	$\boldsymbol{\sigma_{i_0}}(u',u)$	2,588	1,678	3,242	6,900
	$b(i)=1$	$u'=2$	$\boldsymbol{\sigma_{i_1}}(u',u)$	5,854	2,588	-∞	-∞
	$A=min(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			2,588	1,678	-∞	-∞

				$i=8$	$i=7$	$i=6$	$i=5$
$u=2$	$b(i)=0$	$u'=3$	$\boldsymbol{\sigma_{i_0}}(u',u)$	-∞	15,630	3,632	3,948
	$b(i)=1$	$u'=4$	$\boldsymbol{\sigma_{i_1}}(u',u)$	-∞	4,732	1,454	4,804
	$A=min(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			-∞	4,732	1,454	3,948
	$A=max(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			-∞			

				$i=4$	$i=3$	$i=2$	$i=1$
$u=2$	$b(i)=0$	$u'=3$	$\boldsymbol{\sigma_{i_0}}(u',u)$	6,228	3,144	6,228	-∞
	$b(i)=1$	$u'=4$	$\boldsymbol{\sigma_{i_1}}(u',u)$	15,630	5,854	-∞	-∞
	$A=min(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			6,228	3,144	-∞	-∞

				$i=8$	$i=7$	$i=6$	$i=5$
$u=3$	$b(i)=0$	$u'=2$	$\boldsymbol{\sigma_{i_0}}(u',u)$	-∞	-∞	4,803	4,732
	$b(i)=1$	$u'=1$	$\boldsymbol{\sigma_{i_1}}(u',u)$	-∞	-∞	15,630	4,324
	$A=min(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			-∞	-∞	4,803	4,324
	$A=max(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			-∞			

				$i=4$	$i=3$	$i=2$	$i=1$
$u=3$	$b(i)=0$	$u'=2$	$\boldsymbol{\sigma_{i_0}}(u',u)$	3,948	6,228	-∞	-∞
	$b(i)=1$	$u'=1$	$\boldsymbol{\sigma_{i_1}}(u',u)$	0,946	3,242	6,900	15,630
	$A=min(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			0,946	3,242	-∞	-∞

				$i=8$	$i=7$	$i=6$	$i=5$
$u=4$	$b(i)=0$	$u'=4$	$\boldsymbol{\sigma_{i_0}}(u',u)$	-∞	-∞	3,074	3,074
	$b(i)=1$	$u'=3$	$\boldsymbol{\sigma_{i_1}}(u',u)$	-∞	-∞	4,732	2,074
	$A=min(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			-∞	-∞	3,074	2,074
	$A=max(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			-∞			

				$i=4$	$i=3$	$i=2$	$i=1$
$u=4$	$b(i)=0$	$u'=4$	$\boldsymbol{\sigma_{i_0}}(u',u)$	4,804	15,630	-∞	-∞
	$b(i)=1$	$u'=3$	$\boldsymbol{\sigma_{i_1}}(u',u)$	3,994	6,900	15,630	-∞
	$A=min(\boldsymbol{\sigma_{i_0}}(u',u),\boldsymbol{\sigma_{i_1}}(u',u))$			3,994	6,900	-∞	-∞

TABLA 4.13: Comparación de las métricas para obtener la matriz A

Algunos de los valores de las métricas obtenidas son -∞, es decir, son transiciones sin probabilidad de ser producidas. Así que estas métricas no se registran en la matriz A y ésta tiene un total de $\mathcal{L}=20$ valores de métricas diferentes.

La estructura de las $\mathcal{L}=20$ columnas de la matriz A es la siguiente:

$$A[l]=\begin{bmatrix} SIGMA \\ POSTESTADO \\ ANTESTADO \\ TRELLSTEP \end{bmatrix}=\begin{bmatrix} \boldsymbol{\sigma_i}(u',u) \\ u \\ u' \\ i \end{bmatrix}$$

Por tanto, para cada métrica debe guardarse también el estado u, el estado u', y el paso del trellis i en que se produce la transición.

Teniendo en cuenta todo esto, la matriz A queda de la siguiente manera:

	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$
$A[l].\boldsymbol{\sigma_i}(u',u)$	15,630	6,526	3,632	4,732	4,004	1,454	4,804
$A[l].u$	1	1	1	2	1	2	3
$A[l].u'$	2	1	2	4	2	4	2
$A[l].i$	8	8	7	7	6	6	6

	$l=8$	$l=9$	$l=10$	$l=11$	$l=12$	$l=13$	$l=14$
$A[l].\boldsymbol{\sigma_i}(u',u)$	3,074	5,854	3,948	4,324	2,074	2,588	6,228
$A[l].u$	4	1	2	3	4	1	2
$A[l].u'$	4	1	3	1	3	1	3
$A[l].i$	6	5	5	5	5	4	4

	$l=15$	$l=16$	$l=17$	$l=18$	$l=19$	$l=20$
$A[l].\boldsymbol{\sigma_i}(u',u)$	0,946	3,994	1,678	3,144	3,242	6,900
$A[l].u$	3	4	1	2	3	4
$A[l].u'$	1	3	1	3	1	3
$A[l].i$	4	4	3	3	3	3

TABLA 4.14: Matriz A

Una vez obtenida la matriz A se reordenan sus columnas en una nueva **matriz** llamada R cuyas columnas cumplen la siguiente condición para todo $l = 1, 2, 3, ..., 20$:

$$R[l].\boldsymbol{\sigma_i}(u',u) \geq R[l+1].\boldsymbol{\sigma_i}(u',u)$$

	$l=1$	$l=2$	$l=3$	$l=4$	$l=5$	$l=6$	$l=7$
$R[l].\boldsymbol{\sigma_i}(u',u)$	15,630	6,900	6,526	6,228	5,854	4,804	4,732
$R[l].u$	1	4	1	2	1	3	2
$R[l].u'$	2	3	1	3	1	2	4
$R[l].i$	8	3	8	4	5	6	7

	$l=8$	$l=9$	$l=10$	$l=11$	$l=12$	$l=13$	$l=14$
$R[l].\boldsymbol{\sigma_i}(u',u)$	4,324	4,004	3,994	3,948	3,632	3,242	3,144
$R[l].u$	3	1	4	2	1	3	2
$R[l].u'$	1	2	3	3	2	1	3
$R[l].i$	5	6	4	5	7	3	3

	$l=15$	$l=16$	$l=17$	$l=18$	$l=19$	$l=20$
$R[l].\boldsymbol{\sigma_i}(u',u)$	3,074	2,588	2,074	1,678	1,454	0,946
$R[l].u$	4	1	4	1	2	3
$R[l].u'$	4	1	3	1	4	1
$R[l].i$	6	4	5	3	6	4

TABLA 4.15: Matriz R

A continuación se obtiene la **estimación *soft*** de la palabra a la entrada del codificador convolucional más los dos bits de cola ($x'_{R\,soft}$). Es decir, la estimación de $x = [11011101]$. Dicha estimación se obtiene como en el algoritmo Max-Log-MAP:

$$\begin{aligned} L(b_i \mid Y_1^n) &= \max_{\{u',u\}\Rightarrow b_i=+1} \boldsymbol{\sigma_i}(u',u) - \max_{\{u',u\}\Rightarrow b_i=-1} \boldsymbol{\sigma_i}(u',u) \\ &= \max_{\{u',u\}} \boldsymbol{\sigma_{i_1}}(u',u) - \max_{\{u',u\}} \boldsymbol{\sigma_{i_0}}(u',u) \end{aligned} \tag{4.13}$$

Para cada paso del trellis se determina el símbolo $soft$ de salida restando al valor máximo de $\boldsymbol{\sigma_{i_1}}(u',u)$ el valor máximo de $\boldsymbol{\sigma_{i_0}}(u',u)$. Los valores máximos de $\boldsymbol{\sigma_{i_0}}(u',u)$ para cada paso del trellis se muestran en la tabla 4.16. Los valores máximos de $\boldsymbol{\sigma_{i_1}}(u',u)$ para cada paso del trellis se muestran en la tabla 4.17.

$\boldsymbol{\sigma_{i_0}}(u',u)$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$
$u'=1$	6,900	3,242	1,678	2,588	5,854	6,526	6,526	6,526
$u'=2$	-∞	-∞	6,228	3,948	4,732	4,804	-∞	-∞
$u'=3$	-∞	6,228	3,144	6,228	3,948	3,632	15,630	-∞
$u'=4$	-∞	-∞	15,630	4,804	3,074	3,074	-∞	-∞
$\max \boldsymbol{\sigma_{i_0}}$	6,900	6,228	15,630	6,228	5,854	6,526	15,630	6,526

TABLA 4.16: Tabla de los valores máximos de $\boldsymbol{\sigma_{i_0}}(u',u)$

$\boldsymbol{\sigma_{i_1}}(u', u)$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$	$i = 8$
$u' = 1$	15,630	6,900	3,242	0,946	4,324	15,630	-∞	-∞
$u' = 2$	-∞	-∞	2,588	5,854	15,630	4,004	3,632	15,630
$u' = 3$	-∞	15,630	6,900	3,994	2,074	4,732	-∞	-∞
$u' = 4$	-∞	-∞	5,854	15,630	4,804	1,454	4,732	-∞
$\max \boldsymbol{\sigma_{i_1}}$	15,630	15,630	6,900	15,630	15,630	15,630	4,732	15,630

TABLA 4.17: Tabla de los valores máximos de $\boldsymbol{\sigma_{i_1}}(u', u)$

Así que la obtención de la secuencia $x'_{R\,soft}$ es:

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$	$i = 8$
$\max \boldsymbol{\sigma_{i_1}}$	15,630	15,630	6,900	15,630	15,630	15,630	4,732	15,630
$\max \boldsymbol{\sigma_{i_0}}$	6,900	6,228	15,630	6,228	5,854	6,526	15,630	6,526
$x'_{R\,soft}(i)$	8,730	9,402	-8,730	9,402	9,776	9,104	-10,898	9,104

TABLA 4.18: Secuencia con valores de símbolo soft ($x'_{R\,soft}$)

La estimación $soft$ de la secuencia de entrada $x_C = [110111]$, es la de la palabra $x'_{R\,soft}$ sin los últimos dos bits, que han sido añadidos a la entrada del codificador convolucional para que el estado inicial y final del trellis fueran el mismo :

	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
$x_{R\,soft}(i)$	8,730	9,402	-8,730	9,402	9,776	9,104

TABLA 4.19: Estimación $soft$ de la secuencia de entrada x_C

Por último se ejecuta la **subrutina *traceback***. El anterior (u') y posterior (u) estado de cada registro son usados para recuperar la correspondiente secuencia de información de la manera que muestra la figura 4.2. Las operaciones empiezan en el paso de trellis donde el registro de la transición ha sido originalmente insertado en la lista.

La primera secuencia recuperada es la correspondiente a la métrica almacenada en $R(l = 1)$.

1. **Secuencia correspondiente a la métrica almacenada en** $R(l = 1)$

Los valores guardados en la posición $l = 1$ de la matriz R son:

$l = 1$
$R[l = 1].i = 8$
$R[l = 1].\boldsymbol{\sigma_i}(u', u) = 15,630$
$R[l = 1].u' = 2$ / $R[l = 1].u = 1$
$\boldsymbol{b_1(8) = 1}$

(a) Bucle I.

Este bucle no se ejecuta porque $j = i + 1 = 8 + 1 = 9$ es mayor que N ($N = 8$).

(b) Bucle II.

$u = R[l = 1].u' = 2$
$j = R[l = 1].i - 1 = 8 - 1 = 7$

i. *Iteración j=7.*

$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 2))$

$$u' = \left\{ \begin{array}{lllll} \text{si} & b_1(7) = 0 & \Rightarrow & u' = 3 & \Rightarrow \quad \boldsymbol{\sigma_{7_0}}(3, 2) = 15,630 \\ \\ \text{si} & b_1(7) = 1 & \Rightarrow & u' = 4 & \Rightarrow \quad \boldsymbol{\sigma_{7_1}}(4, 2) = 4,732 \end{array} \right\} = 3$$

$\boldsymbol{b_1(7)} = bit(u' = 3, u = 2) = \mathbf{0}$
$u = u' = 3$

ii. *Iteración j=6.*

$j = 7 - 1 = 6$
$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 3))$

$$u' = \left\{ \begin{array}{lllll} \text{si} & b_1(6) = 0 & \Rightarrow & u' = 2 & \Rightarrow \quad \boldsymbol{\sigma_{6_0}}(2, 3) = 4,804 \\ \\ \text{si} & b_1(6) = 1 & \Rightarrow & u' = 1 & \Rightarrow \quad \boldsymbol{\sigma_{6_1}}(1, 3) = 15,630 \end{array} \right\} = 1$$

$\boldsymbol{b_1(6)} = bit(u' = 1, u = 1) = \mathbf{1}$
$u = u' = 1$

iii. *Iteración j=5.*

$j = 6 - 1 = 5$

$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 1))$

$$u' = \left\{ \begin{array}{llllll} \text{si} & b_1(5) = 0 & \Rightarrow & u' = 1 & \Rightarrow & \boldsymbol{\sigma_{5_0}}(1,1) = 5,854 \\ & & & & & \\ \text{si} & b_1(5) = 1 & \Rightarrow & u' = 2 & \Rightarrow & \boldsymbol{\sigma_{5_1}}(2,1) = 15,630 \end{array} \right\} = 2$$

$\boldsymbol{b_1(5)} = bit(u' = 2, u = 1) = \mathbf{1}$
$u = u' = 2$

iv. *Iteración j=4.*
$j = 5 - 1 = 4$
$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 2))$

$$u' = \left\{ \begin{array}{llllll} \text{si} & b_1(4) = 0 & \Rightarrow & u' = 3 & \Rightarrow & \boldsymbol{\sigma_{4_0}}(3,2) = 6,228 \\ & & & & & \\ \text{si} & b_1(4) = 1 & \Rightarrow & u' = 4 & \Rightarrow & \boldsymbol{\sigma_{4_1}}(4,2) = 15,630 \end{array} \right\} = 4$$

$\boldsymbol{b_1(4)} = bit(u' = 4, u = 2) = \mathbf{1}$
$u = u' = 4$

v. *Iteración j=3.*
$j = 4 - 1 = 3$
$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 4))$

$$u' = \left\{ \begin{array}{llllll} \text{si} & b_1(3) = 0 & \Rightarrow & u' = 4 & \Rightarrow & \boldsymbol{\sigma_{3_0}}(4,4) = 15,630 \\ & & & & & \\ \text{si} & b_1(3) = 1 & \Rightarrow & u' = 3 & \Rightarrow & \boldsymbol{\sigma_{3_1}}(3,4) = 6,900 \end{array} \right\} = 4$$

$\boldsymbol{b_1(3)} = bit(u' = 4, u = 4) = \mathbf{0}$
$u = u' = 4$

vi. *Iteración j=2.*
$j = 3 - 1 = 2$
$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 4))$

$$u' = \left\{ \begin{array}{llllll} \text{si} & b_1(2) = 0 & \Rightarrow & u' = 4 & \Rightarrow & \boldsymbol{\sigma_{2_0}}(4,4) = -\infty \\ & & & & & \\ \text{si} & b_1(2) = 1 & \Rightarrow & u' = 3 & \Rightarrow & \boldsymbol{\sigma_{2_1}}(3,4) = 15,630 \end{array} \right\} = 3$$

$\boldsymbol{b_1(2)}= bit(u'=3, u=4) = \mathbf{1}$
$u = u' = 3$

vii. *Iteración j=1.*
$j = 2 - 1 = 1$
$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u=3))$

$$u' = \left\{ \begin{array}{lllll} \text{si} & b_1(1)=0 & \Rightarrow & u'=2 & \Rightarrow \quad \boldsymbol{\sigma_{1_0}}(2,3) = -\infty \\ \\ \text{si} & b_1(1)=1 & \Rightarrow & u'=1 & \Rightarrow \quad \boldsymbol{\sigma_{1_1}}(1,3) = 15,630 \end{array} \right\} = 1$$

$\boldsymbol{b_1(1)}= bit(u'=1, u=3) = \mathbf{1}$
$u = u' = 1$

Por tanto, la posible palabra recibida correspondiente a la métrica almacenada en $R(l=1)$ es:

$b_l(i)$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$
$l=1$	1	1	0	1	1	1	0	1

TABLA 4.20: Secuencia correspondiente a la métrica almacenada en $R(l=1)$

La segunda secuencia recuperada es la correspondiente a la métrica almacenada en $R(l=2)$.

2. **Secuencia correspondiente a la métrica almacenada en** $R(l=2)$

Los valores guardados en la posición $l=2$ de la matriz R son:

$l = 2$
$R[l=2].i = 3$
$R[l=2].\boldsymbol{\sigma_i}(u', u)= 6,900$
$R[l=2].u' = 3$ / $R[l=2].u = 4$
$\boldsymbol{b_2(3) = 1}$

(a) Bucle I.

i. *Iteración j=4.*

$j = 3 + 1 = 4$

$u' = u = 4$

$u = \max_u(\boldsymbol{\sigma_j}(u' = 4, u))$

$$u = \left\{ \begin{array}{llllll} \text{si} & b_2(4) = 0 & \Rightarrow & u = 4 & \Rightarrow & \boldsymbol{\sigma_{4_0}}(4, 4) = 4,804 \\ \\ \text{si} & b_2(4) = 1 & \Rightarrow & u = 2 & \Rightarrow & \boldsymbol{\sigma_{4_1}}(4, 2) = 15,630 \end{array} \right\} = 2$$

$\boldsymbol{b_2(4)} = bit(u' = 4, u = 2) = \mathbf{1}$

ii. *Iteración j=5.*

$j = 4 + 1 = 5$

$u' = u = 2$

$u = \max_u(\boldsymbol{\sigma_j}(u' = 2, u))$

$$u = \left\{ \begin{array}{llllll} \text{si} & b_2(5) = 0 & \Rightarrow & u = 3 & \Rightarrow & \boldsymbol{\sigma_{5_0}}(2, 3) = 4,732 \\ \\ \text{si} & b_2(5) = 1 & \Rightarrow & u = 1 & \Rightarrow & \boldsymbol{\sigma_{5_1}}(2, 1) = 15,630 \end{array} \right\} = 1$$

$\boldsymbol{b_2(5)} = bit(u' = 2, u = 1) = \mathbf{1}$

iii. *Iteración j=6.*

$j = 5 + 1 = 6$

$u' = u = 1$

$u = \max_u(\boldsymbol{\sigma_j}(u' = 1, u))$

$$u = \left\{ \begin{array}{llllll} \text{si} & b_2(6) = 0 & \Rightarrow & u = 1 & \Rightarrow & \boldsymbol{\sigma_{6_0}}(1, 1) = 6,526 \\ \\ \text{si} & b_2(6) = 1 & \Rightarrow & u = 3 & \Rightarrow & \boldsymbol{\sigma_{6_1}}(1, 3) = 15,630 \end{array} \right\} = 3$$

$\boldsymbol{b_2(6)} = bit(u' = 1, u = 3) = \mathbf{1}$

iv. *Iteración j=7.*

$j = 6 + 1 = 7$

$u' = u = 3$

$u = \max_u(\boldsymbol{\sigma_j}(u' = 3, u))$

$$u = \left\{ \begin{array}{llllll} \text{si} & b_2(7) = 0 & \Rightarrow & u = 2 & \Rightarrow & \boldsymbol{\sigma_{7_0}}(3,2) = 15,630 \\ \\ \text{si} & b_2(7) = 1 & \Rightarrow & u = 4 & \Rightarrow & \boldsymbol{\sigma_{7_1}}(3,4) = -\infty \end{array} \right\} = 2$$

$\boldsymbol{b_2(7)} = bit(u' = 3, u = 2) = \mathbf{0}$

v. *Iteración j=8.*
$j = 7 + 1 = 8$
$u' = u = 2$
$u = \max_u(\boldsymbol{\sigma_j}(u' = 2, u))$

$$u = \left\{ \begin{array}{llllll} \text{si} & b_2(8) = 0 & \Rightarrow & u = 3 & \Rightarrow & \boldsymbol{\sigma_{8_0}}(2,3) = -\infty \\ \\ \text{si} & b_2(8) = 1 & \Rightarrow & u = 1 & \Rightarrow & \boldsymbol{\sigma_{8_1}}(2,1) = 15,630 \end{array} \right\} = 3$$

$\boldsymbol{b_2(8)} = bit(u' = 2, u = 3) = \mathbf{1}$

(b) Bucle II.
$u = R[l = 2].u' = 3$
$j = R[l = 2].i - 1 = 3 - 1 = 2$

i. *Iteración j=2.*
$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 3))$

$$u' = \left\{ \begin{array}{llllll} \text{si} & b_2(2) = 0 & \Rightarrow & u' = 2 & \Rightarrow & \boldsymbol{\sigma_{2_0}}(2,3) = -\infty \\ \\ \text{si} & b_2(2) = 1 & \Rightarrow & u' = 1 & \Rightarrow & \boldsymbol{\sigma_{2_1}}(1,3) = 6,900 \end{array} \right\} = 1$$

$\boldsymbol{b_2(2)} = bit(u' = 1, u = 3) = \mathbf{1}$
$u = u' = 1$

ii. *Iteración j=1.*
$j = 2 - 1 = 1$

$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 1))$

$$u' = \left\{ \begin{array}{lllll} \text{si} & b_2(1) = 0 & \Rightarrow & u' = 1 & \Rightarrow \quad \boldsymbol{\sigma_{1_0}}(1,1) = 6,900 \\ \\ \text{si} & b_2(1) = 1 & \Rightarrow & u' = 2 & \Rightarrow \quad \boldsymbol{\sigma_{1_1}}(2,1) = -\infty \end{array} \right\} = 1$$

$\boldsymbol{b_2(1)}$= $bit(u' = 1, u = 1)$ =**0**
$u = u' = 1$

Por tanto, la posible palabra recibida correspondiente a la métrica almacenada en $R(l = 2)$ es:

$b_l(i)$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$	$i = 7$	$i = 8$
$l = 2$	0	1	1	1	1	1	0	1

TABLA 4.21: Secuencia correspondiente a la métrica almacenada en $R(l = 2)$

La tercera secuencia recuperada es la correspondiente a la métrica almacenada en $R(l = 3)$.

3. **Secuencia correspondiente a la métrica almacenada en** $R(l = 3)$

Los valores guardados en la posición $l = 3$ de la matriz R son:

$l = 3$
$R[l = 3].i = 8$
$R[l = 3].\boldsymbol{\sigma_i}(u', u)$= $6,526$
$R[l = 3].u' = 1$ / $R[l = 3].u = 1$
$\boldsymbol{b_3(8) = 0}$

(a) Bucle I.
Este bucle no se ejecuta porque $j = i + 1 = 8 + 1 = 9$ es mayor que N ($N = 8$).

(b) Bucle II.
$u = R[l = 3].u' = 1$
$j = R[l = 3].i - 1 = 8 - 1 = 7$

i. *Iteración j=7.*

$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 1))$

$$u' = \left\{ \begin{array}{lllll} \text{si} \quad b_3(7) = 0 & \Rightarrow & u' = 1 & \Rightarrow & \boldsymbol{\sigma_{7_0}}(1,1) = 6,526 \\ \\ \text{si} \quad b_3(7) = 1 & \Rightarrow & u' = 2 & \Rightarrow & \boldsymbol{\sigma_{7_1}}(2,1) = 3,632 \end{array} \right\} = 1$$

$\boldsymbol{b_3(7)} = bit(u' = 1, u = 1) = \mathbf{0}$

$u = u' = 1$

ii. *Iteración j=6.*

$j = 7 - 1 = 6$

$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 1))$

$$u' = \left\{ \begin{array}{lllll} \text{si} \quad b_3(6) = 0 & \Rightarrow & u' = 1 & \Rightarrow & \boldsymbol{\sigma_{6_0}}(1,1) = 6,526 \\ \\ \text{si} \quad b_3(6) = 1 & \Rightarrow & u' = 2 & \Rightarrow & \boldsymbol{\sigma_{6_1}}(2,1) = 4,004 \end{array} \right\} = 1$$

$\boldsymbol{b_3(6)} = bit(u' = 1, u = 1) = \mathbf{0}$

$u = u' = 1$

iii. *Iteración j=5.*

$j = 6 - 1 = 5$

$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 1))$

$$u' = \left\{ \begin{array}{lllll} \text{si} \quad b_3(5) = 0 & \Rightarrow & u' = 1 & \Rightarrow & \boldsymbol{\sigma_{5_0}}(1,1) = 5,854 \\ \\ \text{si} \quad b_3(5) = 1 & \Rightarrow & u' = 2 & \Rightarrow & \boldsymbol{\sigma_{5_1}}(2,1) = 15,630 \end{array} \right\} = 2$$

$\boldsymbol{b_3(5)} = bit(u' = 2, u = 1) = \mathbf{1}$

$u = u' = 2$

iv. *Iteración j=4.*

$j = 5 - 1 = 4$

$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 2))$

$$u' = \left\{ \begin{array}{lllll} \text{si} \quad b_3(4) = 0 & \Rightarrow & u' = 3 & \Rightarrow & \boldsymbol{\sigma_{4_0}}(3,2) = 6,228 \\ \\ \text{si} \quad b_3(4) = 1 & \Rightarrow & u' = 4 & \Rightarrow & \boldsymbol{\sigma_{4_1}}(4,2) = 15,630 \end{array} \right\} = 4$$

$\boldsymbol{b_3(4)}= bit(u'=4, u=2) =\mathbf{1}$
$u = u' = 4$

v. *Iteración j=3.*
$j = 4 - 1 = 3$
$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 4))$

$$u' = \left\{ \begin{array}{llllll} \text{si} & b_3(3) = 0 & \Rightarrow & u' = 4 & \Rightarrow & \boldsymbol{\sigma_{3_0}}(4,4) = 15,630 \\ \\ \text{si} & b_3(3) = 1 & \Rightarrow & u' = 3 & \Rightarrow & \boldsymbol{\sigma_{3_1}}(3,4) = 6,900 \end{array} \right\} = 4$$

$\boldsymbol{b_3(3)}= bit(u'=4, u=4) =\mathbf{0}$
$u = u' = 4$

vi. *Iteración j=2.*
$j = 3 - 1 = 2$
$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 4))$

$$u' = \left\{ \begin{array}{llllll} \text{si} & b_3(2) = 0 & \Rightarrow & u' = 4 & \Rightarrow & \boldsymbol{\sigma_{2_0}}(4,4) = -\infty \\ \\ \text{si} & b_3(2) = 1 & \Rightarrow & u' = 3 & \Rightarrow & \boldsymbol{\sigma_{2_1}}(3,4) = 15,630 \end{array} \right\} = 3$$

$\boldsymbol{b_3(2)}= bit(u'=3, u=4) =\mathbf{1}$
$u = u' = 3$

vii. *Iteración j=1.*
$j = 2 - 1 = 1$
$u' = \max_{u'}(\boldsymbol{\sigma_j}(u', u = 3))$

$$u' = \left\{ \begin{array}{llllll} \text{si} & b_3(1) = 0 & \Rightarrow & u' = 2 & \Rightarrow & \boldsymbol{\sigma_{1_0}}(2,3) = -\infty \\ \\ \text{si} & b_3(1) = 1 & \Rightarrow & u' = 1 & \Rightarrow & \boldsymbol{\sigma_{1_1}}(1,3) = 15,630 \end{array} \right\} = 1$$

$\boldsymbol{b_3(1)}= bit(u'=1, u=3) =\mathbf{1}$
$u = u' = 1$

Por tanto, la posible palabra recibida correspondiente a la métrica almacenada en $R(l=3)$ es:

$b_l(i)$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$
$l=3$	1	1	0	1	1	0	0	0

TABLA 4.22: Secuencia correspondiente a la métrica almacenada en $R(l=3)$

El resto de las secuencias que se recuperan mediante su correspondiente métrica almacenada en $R(l)$ se calculan de la misma manera. Al final se acaba obteniendo la siguiente lista de secuencias:

$b_l(i)$		$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$	$i=7$	$i=8$
x'_{R1}	$l=1$	1	1	0	1	1	1	0	1
x'_{R2}	$l=2$	0	1	1	1	1	1	0	1
x'_{R3}	$l=3$	1	1	0	1	1	0	0	0
x'_{R4}	$l=4$	1	0	0	0	1	1	0	1
x'_{R5}	$l=5$	1	1	1	1	0	1	0	1
x'_{R6}	$l=6$	1	1	0	0	1	0	0	1
x'_{R7}	$l=7$	1	1	0	1	0	1	1	1
x'_{R8}	$l=8$	1	1	1	1	1	1	1	1
x'_{R9}	$l=9$	1	1	0	0	1	1	0	0
x'_{R10}	$l=10$	1	0	0	1	1	0	0	1
x'_{R11}	$l=11$	1	1	1	0	0	0	0	1
x'_{R12}	$l=12$	1	1	0	1	0	0	1	0
x'_{R13}	$l=13$	0	0	1	0	1	1	0	1
x'_{R14}	$l=14$	0	1	0	1	0	1	0	1
x'_{R15}	$l=15$	1	1	0	0	0	0	1	1
x'_{R16}	$l=16$	1	0	1	0	0	1	0	1
x'_{R17}	$l=17$	1	1	1	0	1	0	1	1
x'_{R18}	$l=18$	0	0	0	0	0	1	0	1
x'_{R19}	$l=19$	1	1	0	0	0	1	1	0
x'_{R20}	$l=20$	1	0	1	1	0	0	0	1

TABLA 4.23: Secuencias correspondientes a las métricas almacenadas en R

Las 20 secuencias retornadas por el algoritmo MLLA son las mismas que las x'_{Rl} anteriores sin los últimos dos bits, que han sido añadidos a la entrada del codificador convolucional para que el estado inicial y final del trellis fueran el mismo:

$b_l(i)$		$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
x_{R1}	$l=1$	1	1	0	1	1	1
x_{R2}	$l=2$	0	1	1	1	1	1
x_{R3}	$l=3$	1	1	0	1	1	0
x_{R4}	$l=4$	1	0	0	0	1	1
x_{R5}	$l=5$	1	1	1	1	0	1
x_{R6}	$l=6$	1	1	0	0	1	0
x_{R7}	$l=7$	1	1	0	1	0	1
x_{R8}	$l=8$	1	1	1	1	1	1
x_{R9}	$l=9$	1	1	0	0	1	1
x_{R10}	$l=10$	1	0	0	1	1	0
x_{R11}	$l=11$	1	1	1	0	0	0
x_{R12}	$l=12$	1	1	0	1	0	0
x_{R13}	$l=13$	0	0	1	0	1	1
x_{R14}	$l=14$	0	1	0	1	0	1
x_{R15}	$l=15$	1	1	0	0	0	0
x_{R16}	$l=16$	1	0	1	0	0	1
x_{R17}	$l=17$	1	1	1	0	1	0
x_{R18}	$l=18$	0	0	0	0	0	1
x_{R19}	$l=19$	1	1	0	0	0	1
x_{R20}	$l=20$	1	0	1	1	0	0

TABLA 4.24: Lista de secuencias obtenidas mediante el algoritmo MLLA

Como puede observarse, la primera palabra de la lista de secuencias retornadas por el algoritmo MLLA, $x_{R_1} = [110111]$, es justamente la palabra situada en la entrada del codificador $x_C = [110111]$.

Por tanto, el algoritmo MLLA acota el espacio total de posibles palabras enviadas ($2^6 = 64$) a una lista menor (20) dentro de la cual se encuentra la palabra realmente enviada.

—5—
IMPLEMENTACIÓN

EN ESTE CAPÍTULO SE EXPLICA LA IMPLEMENTACIÓN QUE SE HA REALIZADO DEL ALGORITMO DE DECODIFICACIÓN MLLA, DESARROLLADO MEDIANTE MATLAB, CON EL FIN DE SER APLICADO A ESQUEMAS DE FINGERPRINTING.

5.1 ESCENARIO A IMPLEMENTAR

EL PRINCIPAL OBJETIVO de este proyecto es aplicar el algoritmo de decodificación MLLA a un esquema de fingerprinting donde haya una coalición entre dos usuarios que generan una nueva copia con un fingerprint diferente a los suyos para que ninguno de los dos pueda ser incriminado.

El esquema del escenario a implementar es el mostrado en la figura 5.1 donde dos usuarios tienen diferentes copias de un contenido identificado con una marca (o fingerprint). Dichas copias son m_1 y m_2. Estas copias son codificadas mediante un **turbo codificador** obteniéndose las respectivas palabras código y_1 y y_2. Una vez obtenidas y_1 y y_2, los dos usuarios realizan un ataque por **confabulación** mediante la media entre y_1 y y_2. Como resultado se obtiene una palabra con un fingerprint distinto al de cualquiera de los dos atacantes, y_{conf}, que es la palabra que se transmite finalmente a través del **canal con ruido AWGN**. A la salida del canal se obtiene dicha palabra con errores producidos por el ruido existente en el canal, $y_{ruidosa}$. Una vez recibida la palabra $y_{ruidosa}$ el **decodificador MLLA** retorna una lista de un determinado tamaño.

Lo que se espera es que dentro de esta lista se encuentre la palabra correspondiente a la copia de alguno de los dos usuarios que ha participado en la coalición (m_1 y m_2) y, si ninguno de estos aparece, que al menos no se culpe a usuarios inocentes.

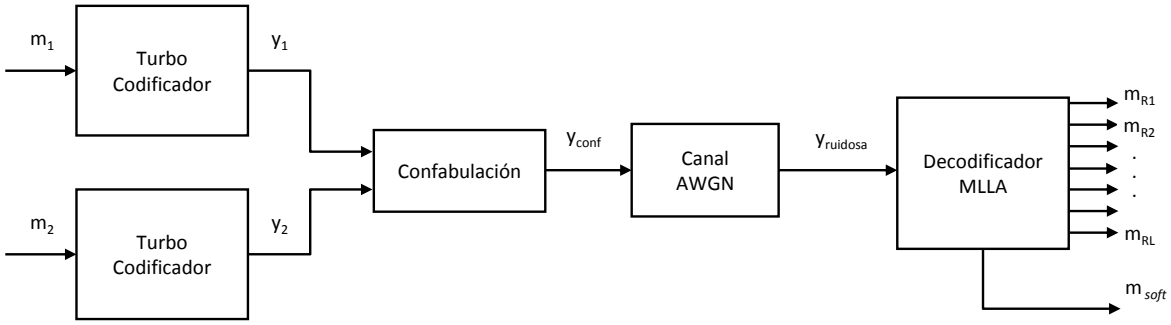

FIGURA 5.1: Esquema del escenario a implementar.

En los siguientes apartados se explica el proceso de implementación, en el orden en que ha sido realizado, que se ha llevado a cabo para ir perfilando cada uno de los bloques que forman parte de dicho esquema.

Cabe señalar que para la realización de este proyecto se ha partido de un código que implementaba un turbo codificador (*turbocod.m*) y un turbo decodificador (*decturbo_cond.m*) .

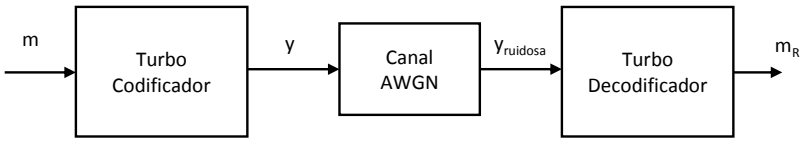

FIGURA 5.2: Esquema del escenario del código de partida.

Así que partiendo de este esquema se ha llegado hasta el de la figura 5.1 realizando todas las modificaciones y adherencias necesarias. A continuación se explican las modificaciones más significativas.

5.2 IMPLEMENTACIÓN DEL ALGORITMO MAX-LOG-MAP

LA PRIMERA MODIFICACIÓN realizada ha sido sobre el código del turbo decodificador (*decturbo_cond.m*), concretamente en la parte referente al algoritmo usado

para realizar la decodificación. Dicho código implementaba el turbo decodificador con el algoritmo MAP y para la realización del decodificador MLLA es necesaria la decodificación mediante el algoritmo Max-Log-MAP, tal y como se indica en el diagrama de la figura 4.1.

Un turbo decodificador, como se muestra en la figura 3.3, está formado por dos decodificadores que intercambian información entre ellos de manera iterativa. El número total de iteraciones que se realizan en esta implementación es 12. Por tanto, el primer y el segundo decodificador intercambiarán información 12 veces. Cada uno de estos decodificadores se implementa dentro del archivo **decturbo_cond.m** mediante una función llamada **maplog.m**.

La cabecera de la función *maplog.m* es la siguiente:

```
function [Le Lu sigma_0 sigma_1]=maplog(y,trellis,Lc,La)
```

Los parámetros que se deben pasar a dicha función son:

- **y:** palabra recibida a través del canal ($y_{ruidosa}$).
- **trellis:** tipo de trellis utilizado para la codificación.
- **Lc:** constante de canal.
- **La:** información a priori intercambiada en cada iteración entre los decodificadores.

Los parámetros retornados por la función son:

- **Le:** estimación *soft* de la palabra m.
- **Lu:** estimación *hard* de la palabra m.
- **sigma_0:** los valores de $\boldsymbol{\sigma_{i_0}}(u', u)$ calculados por el algoritmo Max-Log-MAP.
- **sigma_1:** los valores de $\boldsymbol{\sigma_{i_1}}(u', u)$ calculados por el algoritmo Max-Log-MAP.

Las diferencias entre el algoritmo MAP y el Max-Log-MAP son las expresadas en las ecuaciones que hacen referencia a los valores de $\gamma_i(u', u)$, $\alpha_i(u)$ y $\beta_i(u)$ (3.31, 3.32 y 3.33, respectivamente) y a la obtención del LLR a posteriori (3.34).

El **cálculo de $\gamma_i(u', u)$** en el código original de *maplog.m* estaba realizado en función de $\gamma_{i\_extr}(u', u)$ y la estimación *s*oft y *h*ard de la palabra también, así que se puede obtener $\Gamma_i(u', u)$ haciendo las siguientes aproximaciones:

$$\begin{aligned} \Gamma_i(u', u) &\triangleq \ln(\gamma_i(u', u)) \\ &= ln[Ce^{0.5(L(b_i)b_i + L_c y_{i1} b_i)} e^{\frac{L_c}{2}\sum_{k=2}^{n}(y_{ik}x_{ik})}] \\ &= ln[C]ln[e^{0.5(L(b_i)b_i + L_c y_{i1} b_i)}]ln[\gamma_{i\_extr}(u', u)] \end{aligned} \tag{5.1}$$

Entonces el código relativo a esta variable queda de la siguiente manera:

```
% Cálculo de gamma

for i=1:long
    for j=1:trellis.numStates

        factor_0=exp(0.5*(La(i)*x_0(j,1)+Lc*y_s(i)*x_0(j,1)));
        factor_1=exp(0.5*(La(i)*x_1(j,1)+Lc*y_s(i)*x_1(j,1)));

        % para el cálculo de la gamma extrínseca
        % sólo se usan los bits de paridad.
        gammae_0(j,i)=exp(0.5*Lc*(y_p(i)*x_0(j,2)));
        gammae_1(j,i)=exp(0.5*Lc*(y_p(i)*x_1(j,2)));

        mae_0(j,i)=log(gammae_0(j,i));
        mae_1(j,i)=log(gammae_1(j,i));

        gamma_0(j,i)=factor_0*gammae_0(j,i);
        gamma_1(j,i)=factor_1*gammae_1(j,i);

        ma_0(j,i)=log(gamma_0(j,i));
        ma_1(j,i)=log(gamma_1(j,i));

    end
end
```

Este código crea dos tipos de errores:

- Para valores elevados de *La* (la información a priori), MATLAB no puede calcular el valor de *factor_0* ni *factor_1* puesto que el resultado sería un número de un gran orden.
- Al realizar el cálculo exponencial y luego el logaritmo neperiano, MATLAB realiza aproximaciones que causan errores a la hora del cálculo de $\boldsymbol{\sigma_{i_0}}(u', u)$ y $\boldsymbol{\sigma_{i_1}}(u', u)$ y los consecuentes errores a la hora de implementar el algoritmo MLLA cuando se realizan comparaciones entre los valores obtenidos de dichas variables.

Así que para evitar este tipo de errores se ha aplicado la bien conocida relación

$$y = ln(e^x) \Longrightarrow y = x \tag{5.2}$$

a la expresión 5.1 y, obviando la constante C, se obtiene

$$\begin{aligned} \Gamma_i(u', u) &\triangleq \ln(\gamma_i(u', u)) \\ &= ln[e^{0.5(L(b_i)b_i + L_c y_{i1} b_i)} e^{\frac{L_c}{2}\sum_{k=2}^{n}(y_{ik}x_{ik})}] \\ &= ln[e^{0.5(L(b_i)b_i + L_c y_{i1} b_i) + \frac{L_c}{2}\sum_{k=2}^{n}(y_{ik}x_{ik})}] \\ &= 0.5[L(b_i)b_i + L_c y_{i1} b_i] + 0.5L_c \sum_{k=2}^{n}(y_{ik}x_{ik}) \end{aligned} \tag{5.3}$$

que de esta manera permite reescribir el código correspondiente al cálculo de $\Gamma_i(u', u)$ así:

```
% Cálculo de gamma

for i=1:long
    for j=1:trellis.numStates

        factor_0=0.5*(La(i)*x_0(j,1)+Lc*y_s(i)*x_0(j,1));%
        factor_1=0.5*(La(i)*x_1(j,1)+Lc*y_s(i)*x_1(j,1));%

       % para el cálculo de la gamma extrínseca
       % sólo se usan los bits de paridad.
```

```
        mae_0(j,i)=0.5*Lc*(y_p(i)*x_0(j,2));
        mae_1(j,i)=0.5*Lc*(y_p(i)*x_1(j,2));

        ma_0(j,i)=factor_0+mae_0(j,i);
        ma_1(j,i)=factor_1+mae_1(j,i);

    end
end
```

El **cálculo de la $\alpha_i(u)$** utilizada en el algoritmo Max-Log-MAP se hace a partir de la expresión

$$A_i(u) \triangleq \ln(\alpha_i(u)) = \max_{u'}(A_{i-1}(u') + \Gamma_i(u', u)) \tag{5.4}$$

y es importante que esta aproximación se aplique también a las condiciones iniciales de $A_i(u)$

$$\begin{aligned} A_{i=0}(u=0) &= ln(1) = 0 \\ A_{i=0}(u) &= ln(0) = -\infty \quad si\ u \neq 0 \end{aligned} \tag{5.5}$$

Teniendo en cuenta estas expresiones, el valor de $\alpha_i(u)$ en el algoritmo Max-Log-MAP ($A_i(u)$) se implementa de la siguiente manera:

```
% cálculo de las alfas

% inicialización de alfa
alfa=zeros(trellis.numStates,long+1);
alfa(1,1)=1;
alfa=log(alfa);

for i=1:long
    for state=1:trellis.numStates

        prevstate_0=find(proxState(1,:)==state);
                % búsqueda de los 2 posibles estados que preceden
                % al estado actual cuando el bit de entrada es 0
        prevstate_1=find(proxState(2,:)==state);
                % búsqueda de los 2 posibles estados que preceden
```

```
                % al estado actual cuando el bit de entrada es 1
        alfa(state,i+1)=max(
                ma_0(prevstate_0,i)+alfa(prevstate_0,i),
                ma_1(prevstate_1,i)+alfa(prevstate_1,i)
                );

    end
end
```

Y, finalmente, el **cálculo de la $\beta_i(u)$** utilizada en el algoritmo Max-Log-MAP se hace a partir de la expresión

$$B_{i-1}(u') \triangleq \ln(B_{i-1}(u)) = \max_u(B_i(u) + \Gamma_i(u', u)) \tag{5.6}$$

y, de nuevo, deben aproximarse también las condiciones iniciales de $B_i(u)$

$$\begin{aligned} B_{i=n}(u=0) &= ln(1) = 0 \\ B_{i=n}(u) &= ln(0) = -\infty \quad si\ u \neq 0 \end{aligned} \tag{5.7}$$

Teniendo en cuenta estas expresiones, la implementación de $B_i(u)$ es:

```
% Cálculo de las betas

% inicialización de beta
beta=zeros(trellis.numStates,long);
beta(1,long)=1;
beta=log(beta);

for i=long:-1:2
    for state=1:trellis.numStates
        prevstate_0=proxState(1,state);
                % búsqueda del estado que sucede al estado
                % actual cuando el bit de entrada es 0
        prevstate_1=proxState(2,state);
                % búsqueda del estado que sucede al estado
                % actual cuando el bit de entrada es 1

        beta(state,i-1)=max(
```

```
                ma_0(state,i)+beta(prevstate_0,i),
                ma_1(state,i)+beta(prevstate_1,i)
                );

    end
end
```

Una vez obtenidas $\Gamma_i(u', u)$, $A_i(u)$ y $B_i(u)$, el **cálculo de $\boldsymbol{\sigma_i}(u', u)$**, cuando el bit de entrada es 1 o 0, es:

$$\boldsymbol{\sigma_i}(u', u) \;=\; A_{i-1}(u') + \Gamma_i(u', u) + B_i(u) \tag{5.8}$$

Así que la parte del código donde se han implementado los cálculos de los valores de $\boldsymbol{\sigma_{i_0}}(u', u)$ y $\boldsymbol{\sigma_{i_1}}(u', u)$ es el siguiente:

```
% Cálculo de las sigmas

sigma_0=zeros(trellis.numStates,long);
sigma_0=log(sigma_0);
sigma_1=sigma_0;

tam_alf=size(alfa);
col_alf=tam_alf(1,2);

alfanova=alfa(:,1:col_alf-1);

% alfa(u')+gamma(u',u)+betan(u)
% Ahora se debe encontrar el orden en el trellis:
% valores de prox state son los siguientes estados
% las columnas son los actuales estados

for i=1:trellis.numStates
    sigma_0(i,:)=alfanova(i,:)+ma_0(i,:)+beta(proxState(1,i),:);
    sigma_1(i,:)=alfanova(i,:)+ma_1(i,:)+beta(proxState(2,i),:);
end
```

Finalmente, la **obtención de la LLR** es:

$$L(b_i \mid Y_1^n) = \max_{\{u',u\}\Rightarrow b_i=+1} \boldsymbol{\sigma_{i_1}}(u',u) - \max_{\{u',u\}\Rightarrow b_i=-1} \boldsymbol{\sigma_{i_0}}(u',u) \tag{5.9}$$

```
% Cálculo de la LLR

for i=1:long
    max_0=max(sigma_0(:,i));
    max_1=max(sigma_1(:,i));
    L(i)=max_1-max_0;

    if(max_0==0)
        L(i)=inf;
    end
    if(max_1==0)
        L(i)=-inf;
    end
end

% estimación 'soft'
Le=L;
% estimación 'hard'
Lu=sign(Le);
```

5.3 IMPLEMENTACIÓN DE LA MATRIZ R

EN LA ÚLTIMA ITERACIÓN REALIZADA por el turbo decodificador (iteración 12) y a la salida del decodificador 2, se llama a una nueva función, **maplog_matriz.m**. Esta función es la encargada de crear la matriz donde se guardan las probabilidades de transición a partir de las cuales se crea la lista devuelta por el algoritmo MLLA. Es decir, la función encargada de crear la matriz R.

En dicha función, además de realizarse el cálculo de los valores retornados por el algoritmo Max-Log-MAP de la misma forma que se explica en 5.2, se deben hacer todas las operaciones indicadas en el diagrama 4.1.

Primero se ha implementado el código necesario para obtener la matriz A. Esta matriz es la que contiene el valor de las $\boldsymbol{\sigma_i}(u',u)$ que hay que guardar, el estado

anterior u', el posterior u y el paso del trellis i en el que se ha realizado la transición correspondiente.

Para la realización de esta parte del código hay que tener en cuenta que cuando $k = N$ hay que guardar tanto el valor máximo como el mínimo de $\boldsymbol{\sigma_i}(u', u)$ para un determinado estado u y paso i. A veces estos valores son iguales pero su estado anterior u' es diferente. Para que no se eliminen registros que deben ser almacenados se ha hecho la parte del código referenciada como REPETICIÓN TIPO A.

```
m_i=1;
z=1;
for i=long:-1:1;  % control del paso i del trellis

   for j=1:trellis.numStates      % control del estado u

       % se busca a cada estado u su estado u' anterior
       % (dependiendo de si ha recibido un 1 o 0)
       for m=1:trellis.numStates
           if trellis.nextStates(m,1)==j;
               past_st_0=m;
           end
           if trellis.nextStates(m,2)==j;
               past_st_1=m;
           end
       end

       out=0;
       % caso especial encontrar sigma máxima cuando k=N
       if i==long;
           sigma_min(1,m_i)=max(sigma_0(past_st_0,i),
                                sigma_1(past_st_1,i));

           if sigma_min(1,m_i)==sigma_0(past_st_0,i);
               fut_estat(1,m_i)=proxState(1,past_st_0);
               pres_estat(1,m_i)=past_st_0;
               step(1,m_i)=i;
               % se almacenan: sigma / estado u / estado u' / step

               % REPETICIÓN TIPO A:     si sigma_0 y sigma_1 son
               % iguales cuando k=N hay que guardar el máximo y el
```

```
            % mínimo sin eliminar valores de sigma
            if sigma_min(1,m_i)==sigma_1(past_st_1,i);
                sigma_min(1,m_i)=sigma_0(past_st_0,i);
                fut_estat(1,m_i)=proxState(1,past_st_0);
                pres_estat(1,m_i)=past_st_0;
                step(1,m_i)=i;
                m_i=m_i+1;
                sigma_min(1,m_i)=sigma_1(past_st_1,i);
                fut_estat(1,m_i)=proxState(2,past_st_1);
                pres_estat(1,m_i)=past_st_1;
                step(1,m_i)=i;
                out=1;
            end

        else sigma_min(1,m_i)==sigma_1(past_st_1,i);
            fut_estat(1,m_i)=proxState(2,past_st_1);
            pres_estat(1,m_i)=past_st_1;
            step(1,m_i)=i;
        end

        m_i=m_i+1;

    end

    if out==0;
    % si se ejecuta una REPETICIÓN TIPO A, esta parte del
    % código no es necesaria
    sigma_min(1,m_i)=min(sigma_0(past_st_0,i),
                         sigma_1(past_st_1,i));

    % REPETICIÓN TIPO B      (explicado después del código)

    if sigma_min(1,m_i)==sigma_0(past_st_0,i);
        fut_estat(1,m_i)=proxState(1,past_st_0);
        pres_estat(1,m_i)=past_st_0;
        step(1,m_i)=i;
    else sigma_min(1,m_i)==sigma_1(past_st_1,i);
       fut_estat(1,m_i)=proxState(2,past_st_1);
       pres_estat(1,m_i)=past_st_1;
       step(1,m_i)=i;
    end
```

```
        m_i=m_i+1;
        end
  end
 end
```

Para la realización del código se consideró la posiblidad de que dos valores de $\boldsymbol{\sigma_i}(u', u)$ fueran iguales cuando $k \neq N$ (en el código señalado como REPETICIÓN TIPO B). Como posible solución a dicho problema se optó por implementar un código que añadiera la opción de elegir la transición que tuviera un valor de $A_i(u)$ más grande.

Finalmente, después de implementar esta nueva parte de código, se contempló que sobre un canal AWGN esta situación sólo ocurre cuando dos $\boldsymbol{\sigma_i}(u', u)$ comparadas tienen valor -∞. En este caso específico no importa cual de los dos valores se almacene en la matriz puesto que el camino devuelto por cualquiera de ellos será eliminado posteriormente de la lista por ser un camino con probabilidad nula. Así que el trozo de código añadido se desestimó.

Una vez obtenida la matriz A únicamente hay que ordenar sus columnas, desde el valor mayor de $\boldsymbol{\sigma_i}(u', u)$ hasta el menor, para obtener la matriz R.

```
nou_long=trellis.numStates*(long+1);
% matriu_sigma_min es la matriz A
matriu_sigma_min=[sigma_min; fut_estat; pres_estat; step];

% ordenar la matriz A de sigmas mayores a menores
ceros_nou=zeros(1,nou_long);
a=[matriu_sigma_min;ceros_nou];
b = fliplr( sort(a(1,:)) ); % ordena sigmas de mayor a menor

for k = 1:nou_long
    for i= 1:nou_long
        if (b(k)==a(1,i)) && (a(5,i)==0);
            c(k)=a(2,i); % colocar estado u(futuro) en su sigma
                         % correspondiente
            d(k)=a(3,i); % estado u'(pasado)
            e(k)=a(4,i); % step
            a(5,i)=1; % para controlar si hay valores repetidos
```

```
                    % y así no eliminarlos
                break
            end
        end

end

% matriu_sigma_min_ordenada es la matriz R
matriu_sigma_min_ordenada=[b; c; d; e];
```

5.4 IMPLEMENTACIÓN DE LA FUNCIÓN *traceback*

A CONTINUACIÓN, DESPUÉS DE llamar a la función *maplog_matriz.m*, dentro del código de *decturbo_cond.m*, se llama a la función *traceback.m* tal y como indica la figura 4.1. Dicha función es la encargada de encontrar los caminos más probables a través del trellis a partir de los registros guardados en la matriz R y estos formarán la lista de palabras buscadas.

La cabecera de la función **traceback.m** es:

```
function [bit]=traceback( matriu_sigma_min_ordenada, sigma_0,
                          sigma_1, trellis, max_list, long)
```

Los parámetros que se deben pasar a dicha función son:

- **matriu_sigma_min_ordenada:** matriz R.
- **sigma_0:** todos los valores de $\boldsymbol{\sigma}_{i_0}(u', u)$ calculados por el algoritmo Max-Log-MAP.
- **sigma_1:** todos los valores de $\boldsymbol{\sigma}_{i_1}(u', u)$ calculados por el algoritmo Max-Log-MAP.
- **trellis:** tipo de trellis usado para la codificación.
- **max_list:** máximo número de palabras que pueden formar la lista retornada por el algoritmo MLLA.

- **long:** número de palabras que se desee que formen la lista retornada por el algoritmo MLLA (como máximo puede tener el mismo valor que max_list).

El parámetro retornado por la función, **bit**, es una matriz que contiene la lista de palabras retornadas por el algoritmo MLLA.

Las operaciones que debe realizar la función *traceback.m* se muestran en la figura 4.2. Su implementación global es la mostrada a continuación. Las implementaciones del *BUCLE I* y del *BUCLE II* se explican en los subapartados 5.4.1 y 5.4.2 respectivamente.

```
for l=1:max_list

    fut_st=matriu_sigma_min_ordenada(2,l);  % u
    past_st=matriu_sigma_min_ordenada(3,l);  % u'
    step=matriu_sigma_min_ordenada(4,l);  % i

    if trellis.nextStates(past_st,1)==fut_st;
        bit(l,step)=0;
    elseif trellis.nextStates(past_st,2)==fut_st;
        bit(l,step)=1;
    end

    % BUCLE I (aquí se ejecuta el código explicado en el próximo
    % apartado)

    % BUCLE II( aquí se ejecuta el código explicado más adelante)

end
```

Primero se recupera el registro guardado en la matriz R (y su correspondiente estado futuro, estado pasado y paso del trellis) y se decide si en este paso del trellis el bit enviado ha sido un '0' o un '1'. Después mediante BUCLE I y BUCLE II se determinada la palabra que se devolverá en la posición l de la lista.

Tras ejecutar toda esta función hasta un número de max_list veces, se obtiene una lista de max_list posibles palabras.

5.4.1 IMPLEMENTACIÓN DEL BUCLE I

Este bucle se ejecuta para todos los pasos del trellis mayores al almacenado en la matriz R ($j > i$). El estado futuro almacenado en el anterior paso del trellis (u) pasa a ser el estado pasado (u') del presente paso del trellis (j). Para elegir el estado futuro de la transición presente, se elige el de mayor probabilidad. Es decir, el de mayor valor de $\boldsymbol{\sigma_i}(u', u)$. Una vez determinados u' y u, se guarda el bit que haya provocado dicha transición ('0' o '1') del presente paso del trellis (j).

```
for j=step+1:long;
past_st=fut_st; % u'=u

        if sigma_0(past_st,j)>sigma_1(past_st,j)
                bit(l,j)=0;
        fut_st=trellis.nextStates(past_st,1);
        elseif sigma_0(past_st,j)<sigma_1(past_st,j)
                bit(l,j)=1;
                fut_st=trellis.nextStates(past_st,2);
        end

end
```

5.4.2 IMPLEMENTACIÓN DEL BUCLE II

Antes de iniciar el bucle se recupera el estado anterior (u') del paso guardado en la matriz R (i), y éste es el estado futuro del paso $i - 1$.

Después se inicia el bucle para todos los pasos del trellis menores al almacenado en la matriz R ($j < i$). Para elegir el estado pasado de la transición presente (j), se elige el de mayor probabilidad (mayor $\boldsymbol{\sigma_i}(u', u)$). Una vez determinados u' y u, se guarda el bit que haya provocado dicha transición ('0' o '1') del presente paso del trellis (j).

```
past_st=matriu_sigma_min_ordenada(3,l);
fut_st=past_st; % u=u'
```

```
for j=step-1:-1:1;

        for m=1:trellis.numStates
            if trellis.nextStates(m,1)==fut_st;
                past_st_0=m;
            end
            if trellis.nextStates(m,2)==fut_st;
                past_st_1=m;
            end
        end

        if sigma_0(past_st_0,j)>sigma_1(past_st_1,j);
            bit(1,j)=0;
            fut_st=past_st_0;
        elseif sigma_0(past_st_0,j)<sigma_1(past_st_1,j);
            bit(1,j)=1;
            fut_st=past_st_1;
        end
end
```

5.5 IMPLEMENTACIÓN DE LA OBTENCIÓN DE LISTA FINALMENTE RETORNADA POR EL MLLA

A LA LISTA RETORNADA por la función *decturbo_cond.m* se le deben aplicar 3 modificaciones muy importantes para recibir la lista de palabras definitiva que se desea obtener del algoritmo MLLA. Estas modificaciones son las siguientes:

1. Los bits de cola añadidos por el codificador para que la palabra codificada acabe y empiece en el estado inicial deben ser eliminados.

2. Los registros almacenados en la matriz R que tienen un valor de $\sigma_i(u', u)$ =-∞, son transiciones con una probabilidad nula de producirse, así que deben ser eliminados.

3. Es muy importante tener en cuenta que, en la iteración 12, donde se hace aplica el algoritmo MLLA es en el decodificador 2 y sus bits han pasado por el entrelazador. Por lo tanto, hay que pasar la lista devuelta por el algoritmo

por el de-entrelazador para recibir los bits de cada palabra de la lista en la posición correcta.

Teniendo en cuenta estas modificaciones, el código que se ha implementado después de la llamada a la función *decturbo_cond.m* es el mostrado a continuación.

```
[llr xest sigma_0 sigma_1 bit long matriu_sigma_min_ordenada]=
decturbo_cond(y_ruidosa,2,trellis,inter,10,max_col);

bit=bit(:,1:length(xest)); % se eliminan los bits de cola

% se guardan las palabras válidas de la lista devuelta por el MLLA
% las palabras de sigma=-INF son caminos imposibles en el trellis
tam_mat=size(matriu_sigma_min_ordenada);
long_mat=tam_mat(1,2);
mal_cami=0;

for i=1:long_mat
    if matriu_sigma_min_ordenada(1,i)==-Inf
        mal_cami=mal_cami+1;
    end
end

tam=size(bit);
long_bit=tam(1,1);

max_fil_bit=long_bit-mal_cami;
bit=bit(1:max_fil_bit,:);

% realización de la lista definitiva en función del número de
% palabras deseadas de la lista y de los caminos existentes

if max_list<max_fil_bit+1
    bit=bit(1:max_list,:);
elseif max_list>max_fil_bit
    bit=bit(1:max_fil_bit,:);
end

tam=size(bit);
```

```
max_list_def=tam(1,1);

% en la iteración 12 (que es la última del decoficador)se
% devuelven los bits entrelazados, así que hay que desentrelazar
% para recuperar el orden original.
for i=1:max_list_def
    bit2=bit(i,:);
    bit(i,:)=bit2(deinter);
end
```

—6—
SIMULACIONES Y RESULTADOS

EN ESTE CAPÍTULO SE MUESTRAN LAS SIMULACIONES REALIZADAS PARA LA COMPROBACIÓN DE LA CORRECTA IMPLEMENTACIÓN DEL CÓDIGO DEL ALGORITMO DE DECODIFICACIÓN MAX-LOG-MAP Y DEL MLLA. DESPUÉS SE BUSCAN LOS RESULTADOS OBTENIDOS AL USAR DICHOS ALGORITMOS EN ESQUEMAS DE FINGERPRINTING.

6.1 SIMULACIONES DEL ALGORITMO MAX-LOG-MAP

EN LOS APARTADOS 6.1.1, 6.1.2 y 6.1.3 se explican las simulaciones, y sus correspondientes resultados, relacionadas con la implementación del algoritmo de decodificación Max-Log-MAP.

Los códigos de los bloques que forman parte de estas pruebas no están transcritos, pero se explica el funcionamento de cada uno de ellos y las variables recibidas a su entrada y retornadas a su salida.

6.1.1 FUNCIONAMIENTO DEL ALGORITMO MAX-LOG-MAP

La primera simulación realizada tiene como fin comprobar la correcta implementación del algoritmo Max-Log-MAP y determinar el mínimo umbral de SNR para que no se produzca ningún error en la decodificación.

Esta simulación consiste en **turbo codificar** un mensaje aleatorio de 128 bits (m). El mensaje turbo codificado (y) es enviado través de un **canal AWGN** con un

determinado valor de SNR. El mensaje recibidido a la salida del canal ($y_{ruidosa}$) se **turbo decodifica** obteniéndose una estimación *soft* y una estimación *hard* (m_R) de la secuencia de entrada. Los bloques utilizados para llevar a cabo dicha simulación son los mostrados en la figura 6.1.

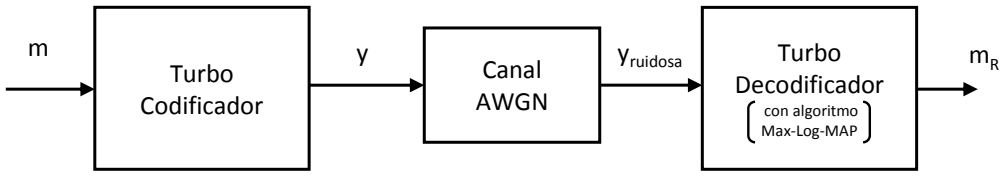

FIGURA 6.1: Simulación del funcionamiento del algoritmo MAx-Log-MAP.

En las primeras pruebas no se ha introducido ningún error de canal y de esta forma se ha comprobado que el turbo decodificador funciona correctamente en ausencia de ruido y devuelve la palabra m a la salida ($m_R = m$).

Tras verificar esto, se ha procedido a introducir el ruido gaussiano. Para ir perfilando el valor de SNR a partir del cual se empiezan a producir errores de decodificación se han realizado simulaciones de 100 iteraciones con SNR=10 dB, SNR=9 dB, SNR=8 dB, ... , SNR=2 dB y SNR=1 dB. Los resultados mostraban que para valores de SNR$\leq$3 dB la introducción de ruido en el canal producía errores a la salida de la turbo decodificación.

Después se han hecho simulaciones de 1000 iteraciones con SNR=3,5 dB, SNR=3,6 dB, SNR=3,7 dB ... SNR=5,9 dB y SNR=6 dB y se ha comprobado que con un valor de SNR igual a 5,2 dB, ya no se produce ningún error al obtener la estimación de la palabra m. Para ratificar esto se ha simulado 3000 veces el esquema de la figura 6.1 para SNR= 5,2 dB obteniendo resultados positivos. Se ha hecho lo mismo con SNR= 5,1 dB y se ha obtenido una probabilidad de error de bit del 1,3%.

Por tanto, para valores de SNR mayores o iguales a 5,2 dB el turbo decodificador implementado mediante el algoritmo Max-Log-MAP devuelve correctamente la palabra turbo codificada a la entrada.

6.1.2 TIEMPO DE EJECUCIÓN

El coste computacional al realizar la turbo decodificación mediante el algoritmo Max-Log-MAP se reduce de manera consiberable respecto a la implementación

hecha mediante el algoritmo MAP. Esto es debido a que los sumatorios se convierten en elección de un valor máximo y las multiplicaciones pasan a ser sumas y estas operaciones reducen el tiempo de ejecución.

Para cuantificar esta mejora de tiempo se han realizado 2 tipos de simulaciones:

- La primera simulación es la mostrada en la figura 6.2, donde una palabra aleatoria de 128 bits (m) se **turbo codifica**, se envía a través de un **canal AWGN** con una relación señal/ruido de 10 dB (se elige este valor porque, como se ha comprobado en 6.1.1, si SNR$\geq$5,2 dB no se produce ningún error en la turbo codificación) y, finalmente, se decodifica mediante un **turbo decodificador** implementado con el algoritmo **MAP**.

  A la palabra obtenida a la salida del turbo decodificador (m_R) se le realiza la **correlación** con la palabra enviada inicialmente (m) para comprobar que el resultado de dicha correlación es R=1. Es decir, para comprobar que la palabra enviada haya sido recibida correctamente ($m = m_R$).

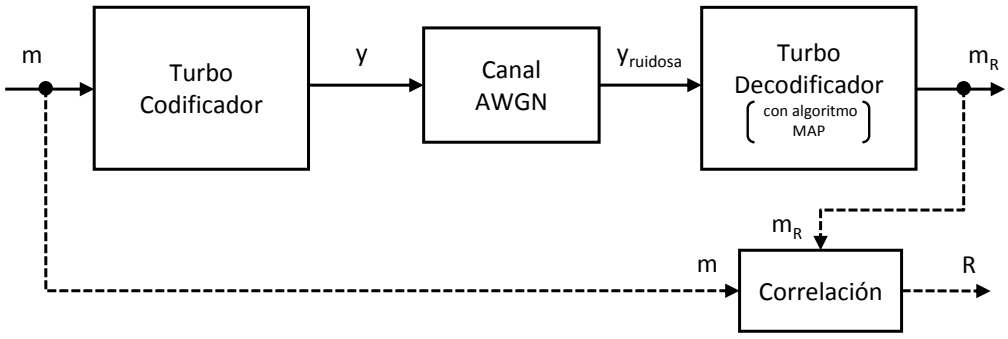

FIGURA 6.2: Esquema de decodificación mediante el algoritmo MAP.

Esta simulación se ha llevado a cabo para 10 palabras distintas y se ha calculado, de manera aproximada, el tiempo de ejecución. Después se ha hecho una media con estos datos del tiempo que tarda este esquema en ser ejecutado.

t_1 (seg)	t_2 (seg)	t_3 (seg)	t_4 (seg)	t_5 (seg)	t_6 (seg)	t_7 (seg)	t_8 (seg)	t_9 (seg)	t_{10} (seg)	t_m (seg)
102,58	112,13	117,27	111,56	121,60	122,38	122,60	118,11	110,01	109,18	114,74

TABLA 6.1: Tiempos de ejecución del algoritmo MAP

Tal y como puede observarse en la tabla 6.1 el tiempo medio de ejecución de

este esquema mediante el algoritmo MAP es, aproximadamente, $t_m = 114,74$ segundos.

- La segunda simulación se muestra en la figura 6.3. Los bloques que forman el esquema de la simulación son los mismos que los de la simulación anterior, con la diferencia de que el turbo decodificador usa el algoritmo de decodificación Max-Log-MAP.

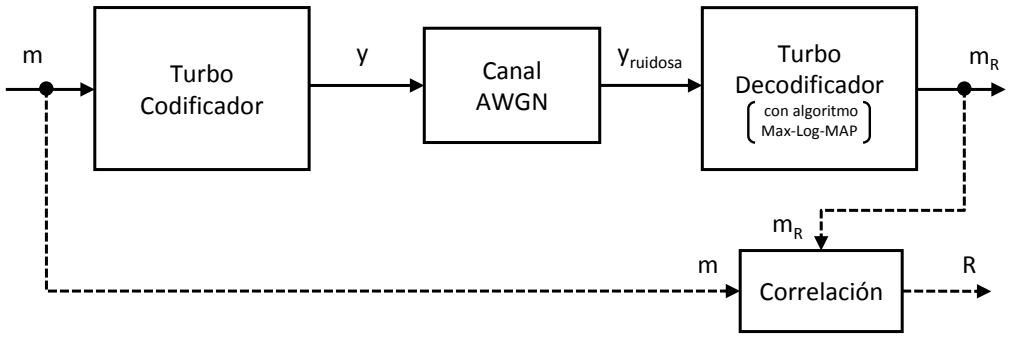

FIGURA 6.3: Esquema de decodificación mediante el algoritmo Max-Log-MAP.

De nuevo, se ha hecho esta simulación para 10 palabras distintas y se ha calculado el tiempo de ejecución de cada una de ellas y el tiempo medio.

t_1 (seg)	t_2 (seg)	t_3 (seg)	t_4 (seg)	t_5 (seg)	t_6 (seg)	t_7 (seg)	t_8 (seg)	t_9 (seg)	t_{10} (seg)	t_m (seg)
17,47	18,03	17,51	18,19	18,16	18,14	18,55	18,35	18,50	18,29	18,12

TABLA 6.2: Tiempos de ejecución del algoritmo Max-Log-MAP

En esta ocasión, el tiempo medio de ejecución del esquema mediante el algoritmo Max-Log-MAP es, aproximadamente, $t_m = 18,12$ segundos.

Por tanto, teniendo en cuenta los dos tipos de simulaciones realizadas, se observa que un turbo decodificador implementado mediante el algoritmo de decodificación Max-Log-MAP es unas 6 veces más rápido que uno implementado mediante el algoritmo de decodificación MAP.

6.1.3 RESISTENCIA A ATAQUES DE CONFABULACIÓN

Las últimas simulaciones relacionadas con el funcionamiento de un turbo decodificador implementado mediante el algoritmo Max-Log-MAP se han realizado para ver el comportamiento de éste cuando dos usuarios realizan una confabulación.

Para la primera de ellas se ha implementado un código que realiza lo que muestra el esquema de la figura 6.4.

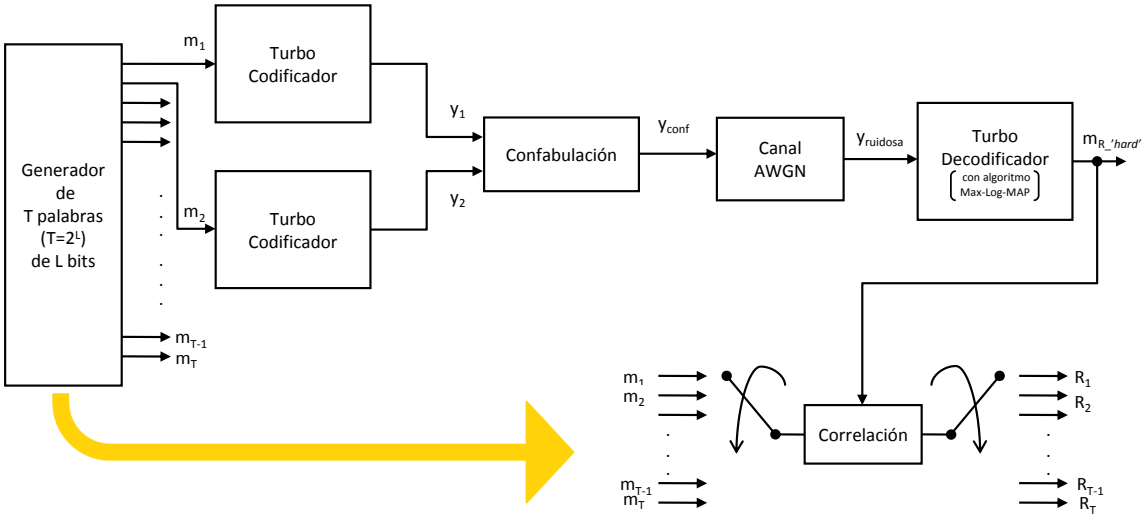

FIGURA 6.4: Simulación del ataque de confabulación I.

Primero se ha hecho un programa que corresponde al **generador de T palabras (T=2^L) de L bits**. Este programa es el encargado de generar una matriz aleatoria de *T (filas) x L (columnas)* que contiene todas las posibles palabras de L bits, ya que T=2^L.

Después se eligen las dos primeras filas de dicha matriz, m_1 y m_2, y se introducen en el **turbo codificador** obteniendo y_1 y y_2 a su salida. Las palabras y_1 y y_2 serán las correspondientes a dos usuarios que pretenden atenuar los fingerprints de sus copias generando una nueva copia que no inculpe a ninguno de ellos. Los dos usuarios realizan un ataque de confabulación, simulado mediante el bloque llamado **confabulación**, que en esta ocasión consiste en la media entre las palabras y_1 y y_2. Como resultado de la confabulación se obtiene la palabra y_{conf}, que es enviada a través de un **canal AWGN** con una relación SNR=10 dB. A la salida del canal se

recibe la palabra $y_{ruidosa}$ y se turbo decodifica. El **turbo decodificador** devuelve la estimación *hard* de la palabra recibida, $m_{R\_'hard'}$.

Luego se compara $m_{R\_'hard'}$ con todas las palabras de la matriz $(m_1, m_2, ..., m_T)$ mediante **correlaciones**. La correspondiente salida de cada una de estas correlaciones es $R_1, R_2, ..., R_T$,. Finalmente se ordenan estas correlaciones de mayor a menor para saber cuáles son las dos palabras de la matriz que más se parecen a la estimación retornada por el turbo decodificador.

Este tipo de simulación se ha realizado con 1000 iteraciones para distintas longitudes de palabra (desde L=6 bits hasta L=128 bits) y siempre han sido R_1 y R_2 las correlaciones de mayor valor. Por tanto, el resultado de dichas simulaciones es que la palabra $m_{R\_'hard'}$ siempre tiene la mayor correlación con las palabras de los usuarios que participan en la confabulación.

En algunas de estas simulaciones la diferencia entre la segunda $(R_{segunda})$ y la tercera $(R_{tercera})$ correlaciones más grandes obtenidas a la salida del bloque correlación no es muy grande (como media, del orden de $R_{segunda} \approx R_{tercera} + 0,2$). Por este motivo se ha hecho esta misma simulación pero esta vez usando la estimación *soft* obtenida a la salida del turbo decodificador $(m_{R\_'soft'})$. Es decir, se ha realizado la simulación mostrada en la figura 6.5.

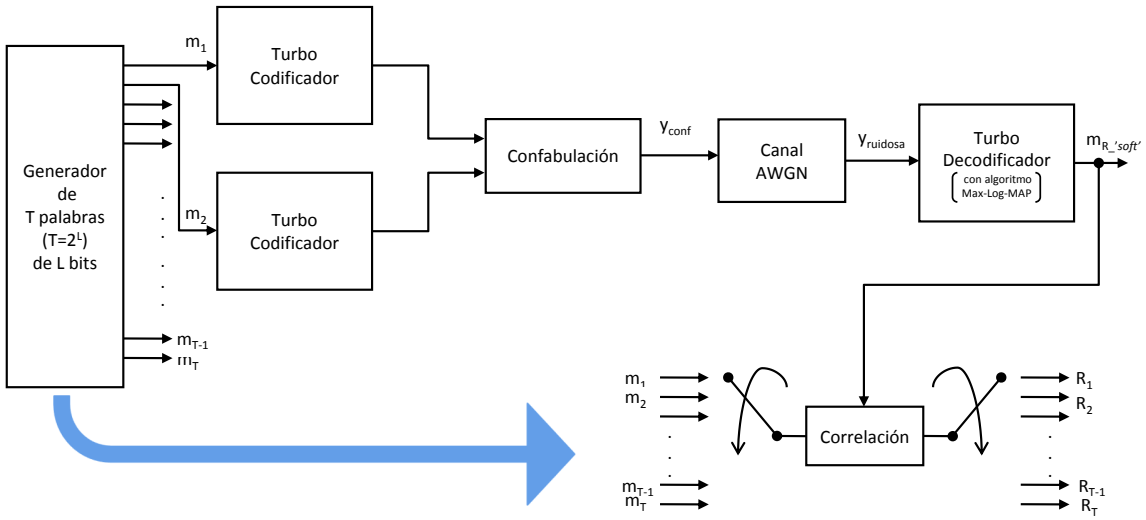

FIGURA 6.5: Simulación del ataque de confabulación II.

Para llevar a cabo dicha simulación se han hecho las correlaciones cruzadas entre $m_{R\_'soft'}$ y todas las palabras de la matriz $(m_1, m_2, ..., m_T)$ de la misma forma que se ha explicado anteriormente. En esta ocasión, R_1 y R_2 siguen siendo las correlaciones de mayor valor. En cambio, la diferencia entre la segunda ($R_{segunda}$) y la tercera ($R_{tercera}$) correlaciones más grandes obtenidas a la salida del bloque correlación es mayor (como media, del orden de $R_{segunda} \approx R_{tercera} + 20$).

En definitiva, las simulaciones realizadas en este apartado muestran que las dos palabras más cercanas a la palabra retornada por el turbo decodificador siempre son las palabras de los dos usuarios que forman parte de la coalición.

6.2 SIMULACIONES DEL ALGORITMO MLLA

EN LOS APARTADOS 6.2.1, 6.2.2, 6.2.3 y 6.2.4 se explican las simulaciones, y sus correspondientes resultados, relacionadas con la implementación del algoritmo de decodificación MLLA.

6.2.1 FUNCIONAMIENTO DEL ALGORITMO MLLA

Mediante la simulación del esquema mostrado en la figura 6.6 se ha probado el correcto funcionamento del algoritmo de decodificación MLLA.

El **generador de 64 palabras de 6 bits** ($64 = 2^6$) crea una matriz aleatoria de *64 (filas) x 6 (columnas)*. Cada una de las filas de la matriz corresponde a una palabra de 6 bits $(m_1, m_2, ..., m_{64})$. Estas palabras se codifican mediante el **codificador convolucional** cuyo trellis añade 2 bits de cola adicionales (para que la palabra empiece y acabe en el estado inicial).

A la salida del codificador se obtienen las palabras $(y_1, y_2, ..., y_{64})$ que tienen una longitud de 16 bits (16 bits= 8 bits de información sistemática + 8 bits de paridad). Estas palabras se guardan dentro de una matriz llamada **matriz C**.

La primera palabra de la matriz C, y_1, es la elegida para ser transmitida a través del **canal AWGN** (cuya relación señal/ruido es SNR=10 dB). La palabra recibida a la salida del canal, $y_{ruidosa}$, se introduce en el **decodificador MLLA**. El decodificador retorna la estimación *soft* de la palabra recibida (m_{soft}) y una lista de 20 estimaciones *hard* $(m_{R1}, m_{R2}, ..., m_{R20},)$.

La lista de palabras retornadas por el decodificador MLLA se codifica mediante un **codificador convolucional** idéntico al primero. A la salida se obtiene la **matriz** $\mathbf{C}_R$ que contiene las 20 palabras codificadas $(y_{R1}, y_{R2}, ..., y_{R20},)$.

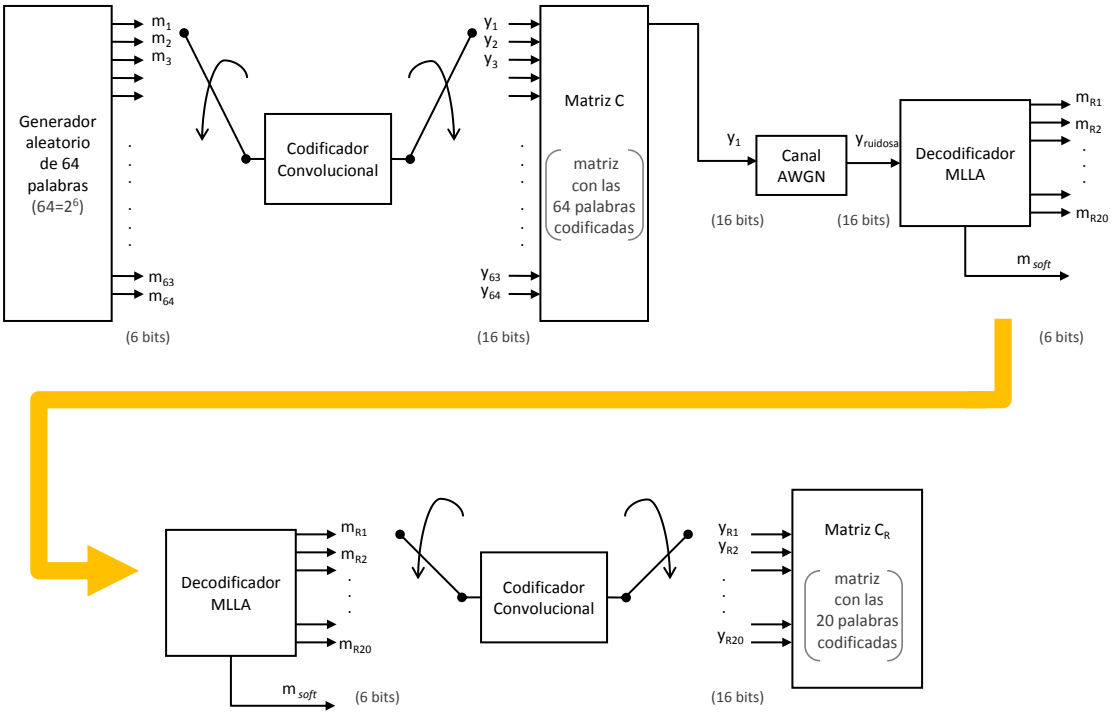

FIGURA 6.6: Funcionamiento del algoritmo MLLA.

Por otro lado, la matriz C se ordena desde las palabras más cercanas a $y_{ruidosa}$ hasta las más lejanas en una nueva matriz llamada $C_{ordenada}$.

Se han obtenido dos resultados de esta simulación que validan la implementación del algoritmo MLLA:

- la primera de las palabras de la lista retornada por el decodificador MLLA (m_{R1}) siempre es la palabra enviada a través del canal. Es decir, siempre coincide con la palabra enviada ($m_{R1} = m_1$).
- entre 8 y 19 palabras de la matriz C_R coinciden, en el mismo orden, con la lista de las 20 primeras palabras de la matriz $C_{ordenada}$.

6.2.2 LONGITUD DE LA LISTA RETORNADA POR EL DECODIFICADOR MLLA

La simulación explicada en este apartado ha sido realizada para ver cómo afecta la longitud de la lista retornada por el decodificador MLLA a las probabilidades de encontrar a cualquiera de los 2 usuarios que forman parte de una coalición. Con este fin se ha implementado un código que realiza lo que muestra el esquema de la figura 6.7.

Primero se han generado dos palabras aleatorias de 128 bits (m_1 y m_2). Estas palabras se codifican a través de dos **turbo codificadores** que están implementados con un trellis que añade 2 bits de cola. A la salida de los turbo codificadores se obtienen las palabras y_1 y y_2 cuya longitud es 390 bits (ya que los turbo codificadores implementados son como el de la figura 3.1 y su razón es de r=$\frac{1}{3}$).

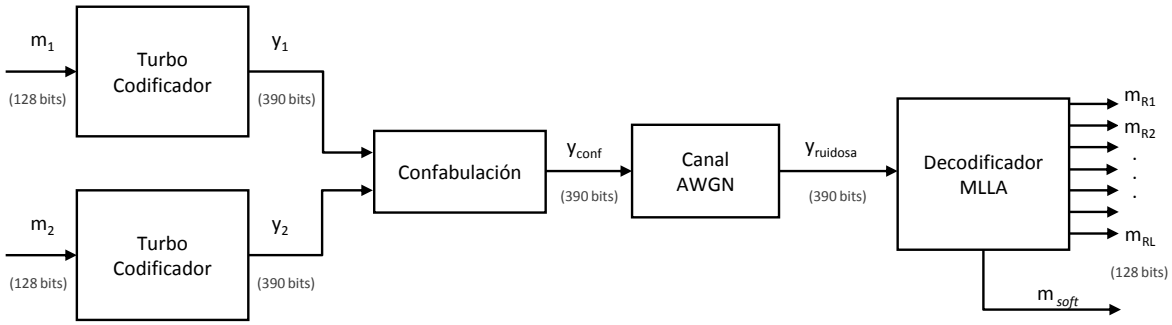

FIGURA 6.7: Rendimiento según el tamaño de la lista.

Las palabras y_1 y y_2 son las correspondientes a dos usuarios que realizan un ataque de confabulación. Dicho ataque se simula mediante el bloque llamado **confabulación**, que en esta ocasión consiste en la media entre las palabras y_1 y y_2. Como resultado de la confabulación se obtiene la palabra y_{conf} que se envía a través de un **canal AWGN** con una relación SNR=10 dB. A la salida del canal se recibe la palabra $y_{ruidosa}$ que se decodifica mediante un turbo decodificador implementado con el algoritmo MLLA.

El **decodificador MLLA** devuelve la estimación *soft* de la palabra recibida (m_{soft}) y una lista de L estimaciones *hard* ($m_{R1}, m_{R2}, ..., m_{RL}$,). Esta simulación se ha hecho 10000 veces y para distintas longitudes (L) de la lista retornada por el decodificador MLLA.

Las probabilidades de encontrar al confabulador 1, al confabulador 2 o a cualquiera de los dos son las mostradas a continuación:

Longitud de lista	Longitud palabra (bits)	Longitud trellis	Número de iteraciones	P encontrar confab 1	P encontrar confab 2	P encontrar ∀ confab
10	128	2	10000	17,69 %	18,18 %	35,87 %
20	128	2	10000	18,36 %	18,10 %	36,46 %
25	128	2	10000	18,03 %	17,86 %	35,89 %
50	128	2	10000	17,58 %	18,67 %	36,25 %
150	128	2	10000	18,24 %	18,82 %	37,06 %
450	128	2	10000	18,35 %	19,01 %	37,36 %

TABLA 6.3: Confabulación con distintas longitudes de lista

Analizando los resultados obtenidos se puede observar que no hay una mejoría considerable, en cuanto a la probabilidad de encontrar a alguno de los usuarios que forman parte de la confabulación, cuando se aumenta la longitud del tamaño de la lista. Para una lista de 10 palabras esta probabilidad es de un 35,87%, y para una lista de 450 esta probabilidad sólo aumenta a un 37,36%. Por tanto, con una lista para el decodificador MLLA de longitud L=10 se tiene, aproximadamente, el mismo rendimiento que para listas de tamaños mayores.

6.2.3 APLICACIÓN DE CRC-16

En este apartado, y en el 6.2.4, se realizan simulaciones para ver cómo afectan diferentes tamaños de trellis (con más estados y más bits de cola) a la probabilidad de encontrar a cualquiera de los 2 usuarios que forman parte de una coalición.

Además, se añade una verificación de redundancia cíclica para averiguar la probabilidad de culpar a un usuario inocente. A continuación se explica en qué consiste este método y la forma en que ha sido implementado:

- **Verificación de redundancia cíclica (CRC)**

  La verificación de redundancia cíclica (abreviado, CRC) es un método de control de integridad de datos de fácil implementación que consiste en la protección de los datos en bloques, denominados tramas. A cada trama se le asigna un segmento de datos denominado código de control. El código CRC contiene datos redundantes con la trama, de manera que los errores no sólo se pueden detectar sino que además se pueden solucionar.

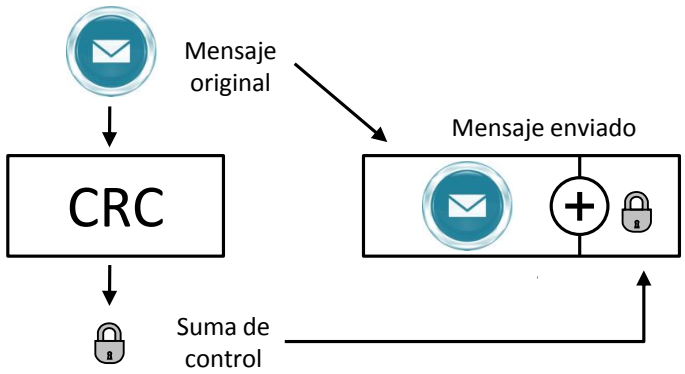

FIGURA 6.8: Funcionamiento del código CRC.

En este proceso de detección de errores, un polinomio predeterminado (denominado polinomio generador y abreviado G(x)) es conocido tanto por el remitente como por el destinatario. El remitente, para comenzar el mecanismo de detección de errores, ejecuta un algoritmo en los bits de la trama, de forma que se genere un CRC, y luego transmite estos dos elementos al destinatario. El destinatario realiza el mismo cálculo a fin de verificar la validez del CRC.

Para el cálculo del CRC se supone que M es el mensaje que corresponde a los bits de la trama que se envía, y que M(x) es el polinomio relacionado. Entonces si M' es el mensaje transmitido (el mensaje inicial al que se concatena un CRC de n bits), el CRC es el siguiente: M'(x)/G(x)=0. Por lo tanto, el código CRC es igual al remanente de la división polinomial de M(x) (al que se le ha anexado los n bits nulos que corresponden a la longitud del CRC) entre G(x).

- **Implementación del CRC-16**

  En este apartado se llevará a cabo la simulación con el CRC-16. Éste tiene como polinomio generador estándar $G(x) = X^{16} + X^{15} + X^2 + 1$. Se ha implementado el siguiente código que es el encargado de generar la palabra resultande de la suma del mensaje incial y el CRC generado:

```
function [transmitida resto] = crc16(h)

    % El polinomio Gx es un  estándar = X^16+X^15+X^2+1
    gx = [1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1];
```

```
    % Se iguala PX
    px = h;

    % Se calcula P(x)x^r
    pxr=[px zeros(1,length(gx)-1)];

    % Se divide pxr entre gx
    % Se obtiene el cociente (c) y el resitudo (r)
    [c r]=deconv(pxr,gx);

    % A veces esta division resulta en valores negativos (-1),
    % así que verificamos que sea 1 o 0
    r=mod(abs(r),2);

    % Se obtiene el crc-16
    resto=r(length(px)+1:end);
    rest=[zeros(1,length(h-1)) resto];
    transmitida=pxr+rest;

    % Cuando el receptor reciba la trama (T(x)) deberá
    % realizar la división T(x)/G(x). Si el resto da 0 la
    % transmisión es correcta. Es decir, debe hacer
    % [transmitida resto]= crc16(T) y resto debe ser 0
    % (resto=0).

end
```

Por tanto, la simulación que se ha hecho ha sido la de la figura 6.9.

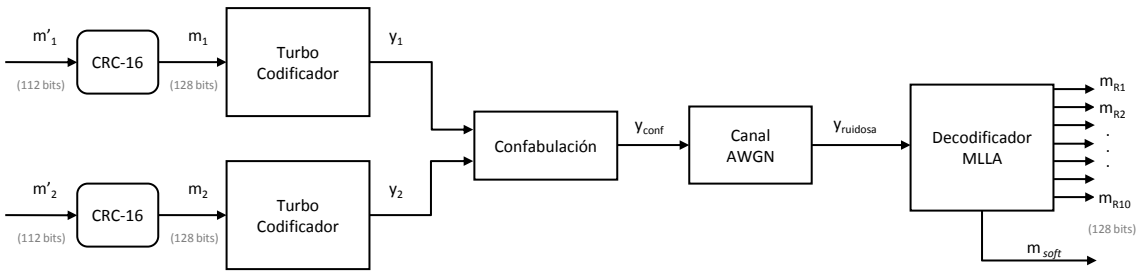

FIGURA 6.9: Resistencia a ataques de confabulación con CRC-16.

Se generan 2 palabras aleatorias de 112 bits (m'_1 y m'_2) a las que se les añaden 16 bits información redundante mediante el **CRC-16**. Así que se obtiene a la salida de cada uno de estos bloques dos palabras de 128 bits (m_1 y m_2). Estas dos palabras son **turbo codificadas** y tras ello se realiza la **confabulación** (mediante su media). La palabra y_{conf} se envía a traves de un **canal AWGN** y se obtiene la palabra $y_{ruidosa}$. La palabra recibida a la salida del canal se decodifica mediante el **decodificador MLLA**. En esta simulación la longitud de la lista retornada por el decodificador ha sido fijada en 10 palabras (tras analizar los resultados obtenidos en 6.2.2).

Esta simulación se ha realizado para trellis de 4, 8, 16, 32 y 64 estados (en la tabla se muestran los bits de cola que añade cada trellis para que la palabra codificada empiece y acabe en el estado inicial). Cada uno de estos tipos de trellis ha sido simulado un total de 10000 veces. Esta vez, además de hallar la probabilidad de encontrar a alguno de los 2 confabuladores, se busca la probabilidad de que haya palabras de la lista retornada por el MLLA que tengan un valor de CRC válido.

Longitud trellis	Longitud de lista	Longitud palabras (bits)	Longitud CRC (bits)	Número de iteraciones	P encontrar confab 1	P encontrar confab 2	P encontrar ∀ confab	P inocentes CRC válido
2	10	112	16	10000	18,50 %	18,59 %	37,09 %	0,011 %
3	10	112	16	10000	13,66 %	13,73 %	27,39 %	0,009 %
4	10	112	16	10000	12,39 %	12,43 %	24,82 %	0,009 %
5	10	112	16	10000	9,15 %	8,99 %	18,14 %	0,007 %
6	10	112	16	10000	6,57 %	7,31 %	13,88 %	0,005 %

TABLA 6.4: Simulación con CRC-16

Tal y como se puede observar en los resultados obtenidos, cuanto mayor es la longitud del trellis usado menor es la probabilidad de encontrar a alguno de los usuarios que forman parte de la coalición atacante. Sin embargo, cuanto mayor es el trellis, menor es la probabilidad de encontrar palabras con CRC válido dentro la lista devuelta por el MLLA que no sean la de alguno de los 2 confabuladores. Es decir, menor es la probabilidad de culpar a usuarios inocentes.

6.2.4 APLICACIÓN DE CRC-64

El último tipo de simulación que se ha hecho ha sido como la implementada en el apartado 6.2.3, pero para tamaños de palabras mayores. En concreto, se ha realizado para palabras de 448 bits a las que se les han añadido 64 bits de CRC.

La **implementación del CRC-64** (teniendo en cuenta que su polimonio generador es $G(x) = X^{64} + X^4 + X^3 + X + 1$) ha sido:

```
function [transmitida resto] = crc64(h)

    %El polinomio Gx es un estándar = X^64 + X^4 + X^3 + X + 1
    gx = [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
    0 0 0 0 1 1 0 1 1];

    px = h;
    pxr=[px zeros(1,length(gx)-1)];
    [c r]=deconv(pxr,gx);
    r=mod(abs(r),2);

    % Se obtiene el crc-64
    resto=r(length(px)+1:end);
    rest=[zeros(1,length(h-1)) resto];
    transmitida=pxr+rest;

end
```

El esquema correspondiente a la turbo codificación de dos palabras de 512 bits (de los cuales, 64 son de redundancia cíclica), m_1 y m_2, y posterior confabulación y decodificación es el siguiente:

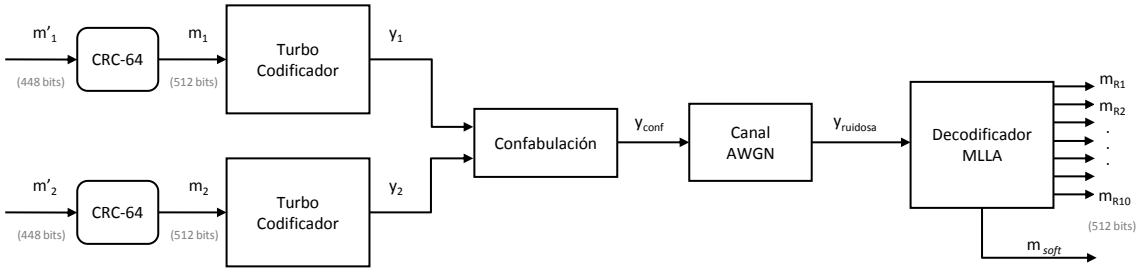

FIGURA 6.10: Resistencia a ataques de confabulación con CRC-64.

Esta simulación se ha realizado para trellis de 4, 8, 16 y 32 estados (en la tabla se muestran los bits de cola que añade cada trellis para que la palabra codificada empiece y acabe en el estado inicial). Cada uno de estos tipos de trellis ha sido simulado un total de 1000 veces (se ha descendido el número de iteraciones porque

para trellis elevados y palabras de 512 bits el tiempo de ejecución de la simulación es muy elevado).

Longitud trellis	Longitud de lista	Longitud palabras (bits)	Longitud CRC (bits)	Número de iteraciones	P encontrar confab 1	P encontrar confab 2	P encontrar ∀ confab	P inocentes CRC válido
2	10	448	64	1000	4,2 %	6,4 %	10,6 %	0 %
3	10	448	64	1000	2,8 %	3,5 %	6,3 %	0 %
4	10	448	64	1000	1,7 %	2,1 %	3,8 %	0 %
5	10	448	64	1000	0,6 %	0,9 %	1,5 %	0 %

TABLA 6.5: Simulación con CRC-64

Como se había analizado anteriormente, cuanto mayor es la longitud del trellis usado menor es la probabilidad de encontrar a alguno de los usuarios que forman parte de la coalición atacante. Además, en estas simulaciones se observa que al aumentar el tamaño de la palabra dicha probabilidad también disminuye de forma considerable. Pero esta vez, aumentando el tamaño de la palabra enviada y el del CRC, se consigue que la probabilidad de encontrar palabras con CRC válido dentro la lista devuelta por el MLLA que no sean la de alguno de los 2 confabuladores sea nula para cualquier longitud del trellis. Es decir, no existe ningún riesgo de culpar a usuarios inocentes.

Por último, en los siguientes gráficos se recogen los datos de las tablas 6.4 y 6.5. A la izquierda se muestran las probabilidades de encontrar a uno o dos de los usuarios que participan en la coalición dentro de la lista de palabras retornadas por el decodificador MLLA. A la derecha se presentan las probabilidades de encontrar a usuarios inocentes con un CRC válido dentro de dicha lista.

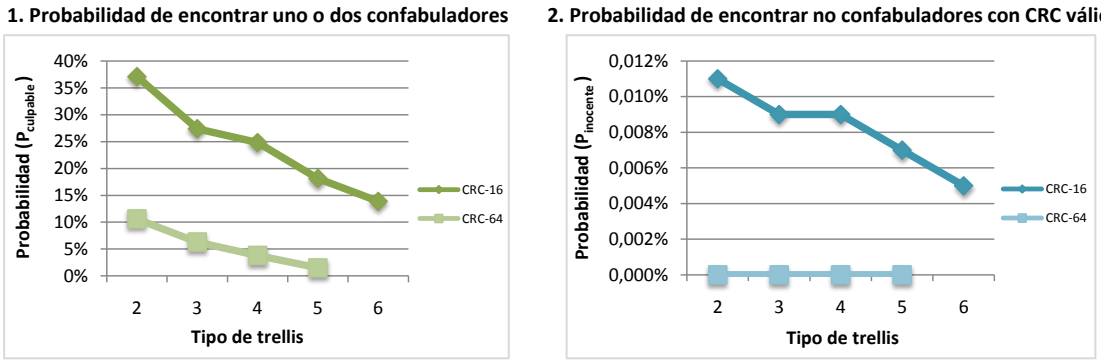

FIGURA 6.11: Gráficos de probabilidades en función del trellis y del CRC usados.

Como se observa en el primer tipo de gráfica, los valores de las probabilidades

de encontrar a alguno de los confabuladores dentro de la lista retornada por el decodificador MLLA no son tan elevados como, *a priori*, se esperaba. Sin embargo, utilizando este algoritmo de decodificación, habrá un determinado porcentaje de veces (equivalente al valor de $P_{culpable}$) en el que podremos encontrar a alguno de los confabuladores sin tener que realizar una decodificación por fuerza bruta.

Además, como se ve en el segundo tipo de gráfica, los valores de las probabilidades de encontrar inocentes en la lista retornada por el MLLA son muy bajos. Éste es un resultado muy importante ya que es imprescindible que usuarios inocentes no resulten inculpados. Como se había fijado desde el inicio, es preferible no encontrar culpables a culpabilizar inocentes.

7

CONCLUSIONES

EN ESTE CAPÍTULO SE EXPLICAN LAS CONCLUSIONES OBTENIDAS AL ESTUDIAR LA POSIBILIDAD DE APLICAR EL ALGORITMO MLLA EN ENTORNOS DE FINGERPRINTING. FINALMENTE, SE APUNTAN LAS LÍNEAS FUTURAS A TRAVÉS DE LAS QUE SE PODRÍA SEGUIR ESTUDIANDO EN ESTE DETERMINADO TIPO DE ENTORNOS.

7.1 Conclusiones y Líneas futuras

El principal objetivo de este proyecto era encontrar una posible solución a los ataques de confabulación que constituyen el principal problema del fingerprinting digital. Para ello se ha optado por la implementación del algoritmo MLLA para ser utilizado en determinados esquemas de fingerprinting.

En una primera etapa, se ha implementado el algoritmo Max-Log-MAP que compone la base del algoritmo MLLA. Para comprobar su correcta implementación se han realizado diversas pruebas. Se ha modificado la relación señal/ruido introducida por el canal AWGN hasta obtener el umbral a partir del cual la decodificación es correcta. El valor de dicho umbral se ha detectado en SNR=5,2 dB. Se ha calculado su tiempo de ejecución comprobando que éste es 6 veces más rápido que el del algoritmo MAP. La última de las simulaciones referentes a este algoritmo se ha realizado para comprobar su funcionamento en esquemas de fingerprinting

donde dos usuarios realizan un ataque por confabulación. Como resultado, el decodificador implementado con el algoritmo Max-Log-MAP siempre retornaba una lista cuyas dos primeras palabras correspondían a los dos usuarios que formaban parte de la coalición.

En la segunda etapa, después de constatar el correcto funcionamiento del algoritmo Max-Log-MAP, se ha implementado el algoritmo MLLA. Se ha verificado su correcto funcionamiento utilizando un codificador/decodificador convolucional y un canal AWGN. Al realizar la correspondiente simulación, la primera palabra de la lista retornada por el algoritmo MLLA, tal y como se deseaba, siempre es la palabra enviada. Tras esto, se ha averiguado que el rendimiento del algoritmo MLLA, en entornos de fingerprinting donde dos usuarios realizan un ataque por confabulación, no tenía una gran mejora si el tamaño de la lista de posibles secuencias de información enviada que es retornada por el mismo aumentaba. Es decir, que con listas de un tamaño pequeño (aproximadamente listas de 10 palabras) se detectaba, con una probabilidad muy parecida a listas de tamaño mayor, a alguno de los confabuladores. Con este mismo esquema de simulación, se ha comprobado que cuanto mayor es el trellis utilizado en turbo codificación y la longitud de la palabra, menor es la probabilidad de encontrar a alguno de los confabuladores dentro de la lista retornada por el algoritmo MLLA. Finalmente se ha encontrado que la probabilidad de culpar a usuarios inocentes, que no han participado en la coalición, introduciendo el método de verificación de redundancia cíclica disminuye cuanto mayor es la longitud del trellis y la longitud de la palabra.

En términos generales, las probabilidades de encontrar a alguno de los usuarios que participan en la coalición no han resultado tener los altos valores que se esperaban (se esperaba una probabilidad mayor del 90%). Así pues, ésta puede ser una primera aproximación que nos acerque a la solución al tipo de problema planteado inicialmente, pero se deben introducir mejoras para tener un mayor rendimiento. Para ello, en un futuro podría concatenarse este tipo de códigos con otro tipo de códigos correctores de errores que lograran una mayor robustez del esquema de codificación frente a ataques de confabulación.

LISTA DE FIGURAS

LISTA DE TABLAS

BIBLIOGRAFÍA

[1] Silvio A. Abrantes, "From BCJR to turbo decoding: MAP algorithms made easier," PhD thesis, University of Kansas, Kansas, 2004.

[2] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *Information Theory, IEEE Transactions on*, vol. 20, no. 2, pp. 284–287, Mar 1974.

[3] S. P. J. A. Erfanian and G. Gulak, "Reduced complexity symbol detectors with parallel structures for isi channels," *IEEE Trans. Commun.*, vol. 42, pp. 1661–1671, 1994.

[4] Matilde Sánchez y Javier Ramos, "Codificación de canal. Turbocodificación," Universidad Carlos III de Madrid, Madrid. [Online]. Available: http://www.tsc.uc3m.es/~mati/Docencia/TC/codificacion_canal.pdf

[5] W. Koch and A. Baier, "Optimum and sub-optimum detection of coded data disturbed by time-varying inter-symbol interference," *IEEE Globecom*, pp. 1679–1684, 1990.

[6] Charan Langton, "Turbo coding and MAP decoding," 2006. [Online]. Available: http://www.complextoreal.com/tutorial.htm

[7] Carl Fredrik Leanderson, "On the design of error control coding for wireless communication systems," PhD thesis, Lund University, Lund, 2002.

[8] C. F. Leanderson and C. -E. W. Sundberg, "List sequence MAP decoding," *IEEE Global Telecommunications Conference*, 2002.

[9] Carl Fredrik Leanderson and Carl-Erik W. Sundberg, "On list sequence turbo decoding," *IEEE Transactions on Communications*, vol. 53, pp. 760–763, 2005.

[10] C. F. Leanderson and C.-E. W. Sundberg, "The max-log list algorithm (MLLA) - a list-sequence decoding algorithm that provides soft-symbol output." *IEEE Transactions on Communications*, vol. 53, no. 3, pp. 433–444, 2005. [Online]. Available: http://dblp.uni-trier.de/db/journals/tcom/tcom53.html#LeandersonS05a

[11] Jorge Castiñeira Moreira and Patrick Guy Farrell, *Essentials of error-control coding*. Chichester, West Sussex PO19 8SQ, England: John Wiley & Sons, Ltd, 2006.

[12] R. Chávez, G. Gamarra y M. Pardo, "Decodificador turbo MAP de varias iteraciones," *Ingeniería y desarrollo*, 2006.

[13] Miguel Rios, "Sistemas de Codificación," , Escuela de Ingeniería Pontificia Universidad Católica de Chile, 2007. [Online]. Available: http://www2.ing.puc.cl/~iee3552/slides10.pdf

[14] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," *Communications, 1995. ICC '95 Seattle, 'Gateway to Globalization', 1995 IEEE International Conference on*, vol. 2, pp. 1009–1013 vol.2, 1995.

[15] Alfonso Francos Romero, "Estudio Teórico de la Arquitectura de Turbo-códigos para aplicaciones de Telefonía Celular de 3G," PhD thesis, Universidad de las Américas Puebla, Puebla, 2007.

[16] S. Martín Ruiz, J. Rodríguez Pedrianes, C. León Hernández, "Códigos correctores de errores. Paralelización de algoritmos exactos con OpenMP y MPI," PhD thesis, Universidad de La Laguna, Santa Cruz de Tenerife, 2007.

[17] Nambirajan Seshadri and Carl-Erik, "List viterbi decoding algorithms with applications," *IEEE Transactions on Communication*, vol. 42, pp. 313–323, 1994.

[18] Joan Tomàs Buliart, "Contribució a la protecció dels drets d'autor i distribució mitjançant la combinació de tècniques de watermarking i codis amb descodificació iterativa amb propietats de rastreig," PhD thesis, Universitat Politècnica de Catalunya, 2010.

[19] Joan Tomàs Buliart and Marcel Fernández and Miguel Soriano, "New considerations about the correct design of turbo fingerprinting codes," *Lecture Notes in Computer Science*, 2008.

[20] J. Vogt and A. Finger, "Improving the max-log-map turbo decoder," *IEE Electronics Letters*, vol. 36, pp. 1937–1939, 2000.

[21] Jason P. Woodard and Lajos Hanzo, "Comparative study of turbo decoding techniques: an overview," *IEEE Transactions on Vehicular Technology*, vol. 49, pp. 2208–2233, 2000.

[22] "Comunicaciones digitales avanzadas," Universidad de Zaragoza, 2006. [Online]. Available: http://diec.unizar.es/asignaturas/defaultWebs/11952/Tema%201%20Convolucional.pdf

[23] "The Wikipedia website," 2010. [Online]. Available: http://en.wikipedia.org/wiki/Turbo_code